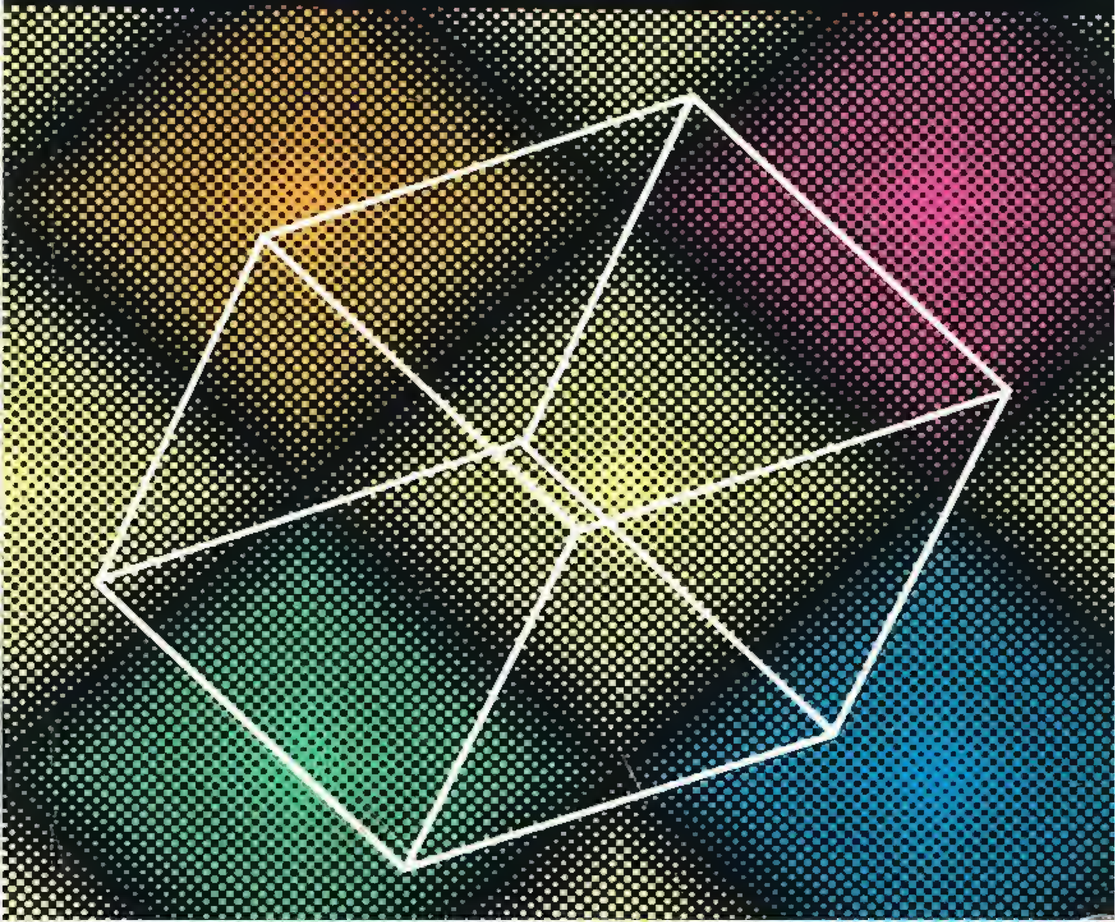
VIC GRAPHICS

NICK HAMPSHIRE



VIC GRAPHICS

NICK HAMPSHIRE

DUCKWORTH

Second impression April 1983
First published in March 1983 by
Gerald Duckworth & Co. Ltd
The Old Piano Factory
43 Gloucester Crescent, London NW1

ISBN 0-7156-1702-8

British Library Cataloguing in Publication Data
Hampshire, Nick
Vic Graphics.
1. Computer graphics
2. VIC (Computer)
I. Title
001.64'43 T385
ISBN 0-7156-1702-8

Typeset by Centrepoint Typesetters Ltd, London
Printed and bound in Great Britain by
Redwood Burn Ltd, Trowbridge, Wilts

CONTENTS

Colour plotting	7
High resolution graphics	29
Graph plotting	87
Using the video memory	99
Scaling and stretching	117
Rotating and moving	137
3D displays	159

AN OVERVIEW

The provision of low cost high resolution colour graphics is probably one of the most exciting and challenging features of a popular home computer like the VIC. With these features a whole new range of exciting applications are opened up for the adventurous programmer. Applications which involve the true visual display of concepts, ideas, and fantasies. In this book I hope to show you how to realise some of the graphics display potential possessed by your machine.

To stimulate your imagination let's first look at some of the possibilities presented by a high resolution colour graphics computer. Perhaps the most obvious application is in simulations, and the most obvious use of simulations is in education. There is an old saying when trying to explain a concept, that a picture is worth a thousand words. This is particularly true in all science related subjects. Relationships can be shown between two or more mathematical functions displayed as curves on the screen, or a mathematical process such as differentiation can be shown graphically taking place. In chemistry three dimensional graphics can be used to show molecular structures and bonding. A chemical process can be displayed and the various reactions simulated by the computer.

Some of the best examples of simulations involving high resolution colour computer graphics come from physics. The teacher has the ability to display the concepts of mechanics, such as Newton's laws, the trajectory of a missile or planetary motion. Magnetic and electrostatic fields and their interrelationship can easily be displayed, as can the path of light through optical systems. In electronics the computer can be used to simulate a circuit and the high resolution graphics used to display the circuit on the screen.

Computer games — which are in the majority of cases just a special fun form of simulation — are obvious candidates for improvement by the use of high resolution colour graphics displays. Although the VIC still cannot match the incredible real time and very realistic displays found on many of the best arcade games, the quality of the home computer's graphics does allow for the programming of some fantastic display based games. In all these games programs the graphics display is augmented by the sound generation ability of the VIC. The range of computer games is enormous, ranging from arcade games like Space

Invaders or Packman to chess programs with a high quality display of a chess board and all the pieces, or the fantasy games like Adventure which can be endowed with some very interesting graphics.

Computer art is an application for high resolution colour computer graphics in which a growing number of people are becoming interested. The artist uses the graphics display as a canvas on which the picture or design is drawn either in a single colour or using all the colours available on the computer. The picture is created by using either a specially written program and an input data base to generate the display, or a light pen or joystick to interactively paint the picture on the screen, much as one would using a paint brush. Such dislays could of course be either a static one off picture or an animated sequence. The generation of animated computer art displays is a subject of increasing interest to creators of cartoon films; this should be within the capabilities of a home computer like the VIC. An example of such graphics was shown in the film 'Star Wars' in the scene where the rebel pilots are briefed on the workings of the 'Death Star'. Full length feature animated films generated by computer can be expected within the next year.

An important application for graphics simulation is using three dimensional graphics software to aid the designing of buildings or engineering structures. This is known as CAD, or Computer Aided Design, and although in commercial applications confined to very large fast computers it is quite possible to perform most of the CAD operations on a machine like the VIC. The designer builds up a model in the computer memory, and can using this data base view the structure from any angle or even go inside. Perspective, light and dark shading, surface texture and colour of solids can all be emulated by such software; some examples of routines to do these functions are given in the last section of this book. Another variation of this type of application is used in flight simulators, where the computer using a previously entered data base, creates a simulated display of a piece of terrain or an airfield as the person using the simulator would see it from any position in three dimensional space. In a flight simulator the position of viewing would depend on how the 'pilot' moved his controls. Simulated landings and take-offs can thus give a visual feedback to the pilot through the use of such computer graphics.

COLOUR PLOTTING

COLOUR CONTROL

The colours which can be displayed on the VIC are divided into two groups. The first group has eight colours, these can be used for the foreground or video colour, RAM stored colour and the border. The second group has 16 colours which can be used for the background colour or for the auxiliary colour in the 'Multicolour' mode. The colours available in each of the two groups are as follows:

AUXILIARY/BACKGROUND		BORDER/CHARACTER
0	Black	Black
1	White	White
2	Red	Red
3	Cyan	Cyan
4	Magenta	Magenta
5	Green	Green
6	Blue	Blue
7	Yellow	Yellow
8	Orange	
9	Light orange	
10	Pink	
11	Light cyan	
12	Light magenta	
13	Light green	
14	Light blue	
15	Light yellow	

Text and graphics displays can be generated in these colours using the colour codes within the print statements. This is adequate for the setting of colours in most displays, but when dealing with high resolution or multicolour displays the commands in the Super Expander are essential for easy programming. It should be remembered that colours can only be defined for single character spaces rather than single display points.

THEORY OF COLOUR PLOTTING

The VIC has two modes of colour operation, 'High resolution' mode and 'Multicolour' mode. The operating mode employed and the colours used are determined by the contents of control registers #15 and #16 of the 6561 and the colour video RAM. The colour video RAM is located in a 506 byte block of memory starting at location $9600 (decimal 38400). If there is more than 8K of user memory, the starting location of colour RAM moves down to $9400 (decimal 37888). The colour video RAM is only four bits wide; bits 0-2 are used to select the character colour and bit 3 is used to determine if that character is in 'High resolution' or 'Multicolour' mode.

The 'High resolution' mode is selected by having bit 3 of the video colour RAM set to zero; this is the normal mode of operation. In this mode there is a one to one correspondence between character generator bits and the dots displayed on the screen. This means that all 'one' bits will be displayed as dots of one colour and all 'zero' bits as dots of another colour. Each character has two colours, a foreground (all the 'one' bits) and a background colour (all the 'zero' bits). One of these colours is determined by the first three bits of the video colour RAM and the other by bits 4-7 of control register # 16.

In normal operation the foreground colour is stored in the video colour RAM and the background colour, which is common to all characters displayed on the screen, is stored in register # 16. This can be reversed so that all characters have the same foreground colour, which is determined by register # 16, and different background colours set by the contents of the colour video RAM. Whether a common foreground or a common background is selected depends on the contents of bit 3 of control register # 16. If bit 3 is set to 1 then the display will have different colour characters on a common background colour; if bit 3 = 0 then all characters will have the same colour against a different colour background. In addition to the foreground and background colours the 6561 allows the colour of the border around the display area to be changed; this is selected by bits 0-2 of control register # 16.

In summary; in 'High resolution' mode the colours used for a particular character are set by:

1) Set bit 3 of register # 16 for common background or common foreground.

```
common foreground – POKE 36879,PEEK(36879) AND 247
common background – POKE 36879,PEEK(36879) OR 8
```

2) Set the common background/foreground colour in bits 4-7 of control register # 16. There are 16 possible colours and it is the colour number as shown in the above table which is stored in the register, as in the following example where variable C is the colour and is set to a value between 0 and 15:

```
POKE 36879,PEEK(36879) AND 15
POKE 36879,PEEK(36879) OR (C*16)
```

return to normal with – POKE 36879,27

3) Set the border colour in bits 0-2 of control register # 16. There are eight possible border colours and it is the colour number

shown in the above table which is stored in the register, as in the following example where variable C is the colour and is set to a value between 0 and 7:

```
POKE 36879, PEEK(36879) AND 248
POKE 36879, PEEK(36879) OR C
```

4) Put the colour code for each character to be displayed into the corresponding location in the colour video RAM. There are eight possible character colours (see above table) and they are stored in bits 0-2 of the 506 locations in the colour video RAM. This is done automatically in a PRINT statement where the character colours can be embedded in the string as colour commands, but if POKE commands are used to put characters into the video RAM then the colour code must also be POKEd into the corresponding location in the colour RAM. Given the column number — COL, and line number — LIN, of the display plus the ASCII code of the character — A, and the colour code for that character — C, the following routine will put the character and its colour into the correct locations in the two video RAMs:

```
100 Q=LIN*22+COL
110 POKE 38400+Q,C
120 POKE 7680+Q, A
```

The 'Multicolour' mode is selected by having bit 3 of the video colour RAM set to one. In this mode there is a two to one correspondence between character generator bits and the dots displayed on the screen. This means that two bits of the character generator matrix for that character code correspond to one dot on the screen, and the colour of that dot is determined by the two bit code in the character generator.

Unlike the 'High resolution' mode in which only two colours can be displayed for each character, 'Multicolour' mode allows four colours per character. However, since two bits of character generator data correspond to a single dot on the screen the horizontal resolution is half that of the 'High resolution' mode. That is, each 8 × 8 character cell in memory maps onto an 8 × 4 character on screen (8 lines of 4 dots). Each character occupies the same space in either mode since both modes can be intermixed in a display; this means that a single dot in 'Multicolour' mode occupies the same space as two horizontal dot positions in the 'High resolution' mode. The amount of memory required for storage of the 8 × 4 'multicolour' characters is the same as that required for the 8 × 8 characters; the data is simply mapped differently on screen.

The 'Multicolour' mode is not suitable for use with the ROM-based character generators but can be very effective when used with a user definable RAM character generator. This is because the ROM character generators are designed for 'High resolution' mode displays where each bit in the character matrix represents a dot position on the screen. In 'Multicolour' mode the character generator contains the colour of each dot by using two bits to represent each display dot; with a ROM character generator most characters will thus appear as an array of different coloured points rather than a character. See the section on high resolution for information on the user of user definable RAM character generators and high resolution point plotting.

In 'Multicolour' mode the two bits of the character generator character matrix which represent each screen dot select one of four colours for that dot. The four codes created by these two bits tell the 6561 where to find the colour information for the dot. The two bit code is not itself a colour code; it is simply a pointer to four different colour codes; this gives more flexibility as each code pointed to has either 3 or 4 bit resolution. The use of a simple two bit pointer, combined with bit 3 of the colour video RAM being used to determine the colour display mode means that it is possible to freely intermix 'High resolution' and 'Multicolour' characters in a display. The colour of the dot can be the background colour, the foreground colour, the exterior border colour or a special auxiliary colour, information on which is stored in bits 4-7 of control register #15. The 'Multicolour' mode select codes are:

0 0 — Background colour
0 1 — Exterior border colour
1 0 — Foreground colour
1 1 — Auxiliary colour

The use of the 'Multicolour' mode can be summarised using the following example:

1) Set the background colour to one of 16 colours; this colour code is stored in the following example in variable C which will have a value between 0 and 15:

```
POKE 36879,PEEK(36879) AND 15
POKE 36879,PEEK(36879) OR (C*16)
```

2) Set the exterior border colour to one of eight colours; this colour code will have a value between 0 and 7 and in the following exam-

ple is stored in variable C:

```
POKE 36879.PEEK(36879) AND 248
POKE 36879,PEEK(36879) OR C
```

3) Set the foreground colour to one of eight colours by POKEing the colour code into the colour video RAM location corresponding to the location of the displayed 'Multicolour' character. Since it is bit 3 of the colour video RAM which determines whether a character is displayed in 'High resolution' or 'Multicolour' mode, 8 should be added to the colour code values for all characters to be displayed in 'Multicolour' mode.

4) Set the auxiliary colour code to one of 16 colours; this colour code will have a value between 0 and 15 and in the following example is stored in variable C:

```
POKE 36878,PEEK(36878) AND 15
POKE 36878,PEEK(36878) OR (C*16)
```

Note: bit 3 of control register #16 has no function in 'Multicolour' mode but should be set to the normal value of 1, unless otherwise required when intermixing both colour display modes.

5) Set up the character generator matrix for each character to be displayed, thus:

	bit									
byte	7	6	5	4	3	2	1	0	**Hex**	**Location**
0	0	0	0	1	1	0	1	1	1B	5120
1	0	0	0	1	1	0	1	1	1B	5121
2	0	0	0	1	1	0	1	1	1B	5122
3	0	0	0	1	1	0	1	1	1B	5123
4	0	0	0	0	0	0	0	0	00	5124
5	0	1	0	1	0	1	0	1	55	5125
6	1	0	1	0	1	0	1	0	AA	5126
7	1	1	1	1	1	1	1	1	FF	5127

This example is for a character in a user definable character generator starting at location 5120. The character has a code value of 0 and shows each of the four colours available in multicolour mode characters thus:

byte	7	6	5	4	3	2	1	0	Hex	Location
0	0	0	0	1	1	0	1	1	1B	5120
1	0	0	0	1	1	0	1	1	1B	5121
2	0	0	0	1	1	0	1	1	1B	5122
3	0	0	0	1	1	0	1	1	1B	5123
4	0	0	0	0	0	0	0	0	00	5124
5	0	1	0	1	0	1	0	1	55	5125
6	1	0	1	0	1	0	1	0	AA	5126
7	1	1	1	1	1	1	1	1	FF	5127

RANDOM COLOURS

DESCRIPTION

Colours can be used to fill blocks of the screen thereby generating interesting effects. This program shows how the colour command can be used to generate colourful dynamically moving patterns. The display consists of a dynamically moving point at which are plotted squares of different colours, the movement of the point and the colour selection are random. The resulting display is a changing pattern of variously shaped different coloured blocks.

RUNNING THE PROGRAM

Since no parameters are input by the program, simply type RUN and watch the program display a constantly changing coloured pattern.

PROGRAM STRUCTURE

110	set starting point on screen
130	set random variables for colour and number of characters of same colour
220	set random variable for movement direction
230-260	move in one of four directions
270-310	check character position is within screen boundary
330-380	plot coloured square

```
1 REM RANDOM COLOURS
2 REM ************************
3 REM
10 REM THE ROUTINE GENERATES A DYNAMICALY
20 REM MOVING COLOUR DISPLAY.
25 SCNCLR
30 REM
40 REM SET COLOUR
45 REM
50 GRAPHIC 2:COLOR 1,1,1,0
54 POKE36879,PEEK(36879)+8
55 REM
100 REM SET CONSTANTS
110 A=10:B=10
120 REM
130 REM RANDOMISE COLOUR
150 REM
160 C=INT(RND(1)*8)
165 IF C=1 THEN 160
170 N=INT(RND(1)*10)
180 REM
190 REM MAIN CHARACTER PLOTTING ROUTINE
200 REM
210 FOR X=0 TO N
220 D=INT(RND(1)*4)
230 IF D=0 THEN A=A+1
240 IF D=1 THEN A=A-1
250 IF D=2 THEN B=B+1
260 IF D=3 THEN B=B-1
265 REM
270 REM WITHIN BOUNDS?
275 REM
280 IF B>18 THEN B=18
290 IF B<1 THEN B=1
300 IF A>18 THEN A=18
310 IF A<1 THEN A=1
320 REM
330 REM PLOT COLOURED CHARACTER
340 REM
350 COLOR 1,1,C,0
360 CHAR A,B," "
370 NEXT X
380 GOTO 160

READY.
```

MAP

DESCRIPTION

Background colours can very effectively be used to fill blocks of the screen with different colours to define outline shapes. High resolution or character plotting can then be used to put details on the outline. This program shows how this can be done and to illustrate the technique draws a map of North America with appropriate text legends. The background colours are set by POKEing the correct colour value into the colour memory In this program only two outline colours are used and for this reason the plotting is divided into two sections, one for each colour. The display is built up from lines or single characters of colour. The data for each line displayed is stored as data statements and consists of sets of three values — line number, column number and number of characters from that position to be plotted continuously on the line. If the display was to be plotted in many different colours then an extra colour parameter should be added to the data tables.

RUNNING THE PROGRAM

Since no parameters are input by the program simply type RUN and watch the program display a map of North America on the screen using different colours for each country.

PROGRAM STRUCTURE

100-140	fill the screen with cyan colour to act as a background to the display
200-280	plot the map of USA in green using data from table lines 310-370
300-370	data table for drawing map of USA, note that the data is stored as a sequence of three values: line, column and length of block
500-570	plot the map of Mexico and Canada in white using data from the table in lines 600-700
600-700	data table for plotting Mexico and Canada
900-1070	put legends on map — note: make sure that the paper colour for the text or high resolution is identical to that of the background colour already plotted

```
1 REM MAP
2 REM ************************
3 REM
10 REM THIS PROGRAM DRAWS A COLOURED
15 REM MAP OF NORTH AMERICA.
20 REM
30 REM
100 REM SET MAP BACKGROUND COLOUR AS BLUE
110 REM
130 COLOR 6,6,0,0
140 PRINT"♥"
165 REM
200 REM DRAW THE U.S.A. IN GREEN
205 REM
210 READ R,S,L
220 IF R=100 THEN GOTO 500
230 P=38400+R*22+S
235 X=7680+R*22+S
240 FOR Q=0 TO L-1
250 POKE P+Q,5
255 POKE X+Q,160
260 NEXT Q
270 REM
280 GOTO 210
290 REM
300 REM DATA FOR PLOTTING U.S.A.
305 REM
310 DATA 6,1,4
320 DATA 7,1,14,7,18,2,8,0,14,8,17,3
330 DATA 9,0,15,9,16,4,10,0,19
340 DATA 11,0,18,12,0,18,13,1,17,14,1,17
350 DATA 15,2,15,16,6,11,17,7,8,17,15,2
360 DATA 18,8,5,18,16,2,19,17,1
370 DATA 100,100,100
495 REM
500 REM DRAW CANADA AND MEXICO IN WHITE
505 REM
510 READ R,S,L
520 IF R=100 THEN 900
530 P=38400+R*22+S
535 X=7680+R*22+S
540 FOR Q=0 TO L-1
550 POKE P+Q,1
555 POKE X+Q,160
560 NEXT Q
570 GOTO 510
595 REM
600 REM DATA FOR CANADA AND MEXICO
605 REM
610 DATA 0,0,20,1,0,20
620 DATA 2,0,20,3,0,20,4,0,20,5,1,19,6,5,15
```

```
623 DATA 7,15,5,8,16,3,9,17,1,16,2,4,17,2,1
626 DATA 17,4,3,18,3,1,18,5,5,19,5,8,20,5,8
630 DATA 21,5,8
700 DATA 100,100,100
895 REM
900 REM PUT NAMES ON MAP
905 REM
910 FOR I=1 TO 3
920 READ R,S,X$
930 X=7680+22*R+S
950 FOR Q=0 TO LEN(X$)-1
970 POKE X+Q,ASC(MID$(X$,Q+1,1))+64
980 NEXT Q
990 NEXT I
1000 GET A$:IF A$="" THEN 1000
1010 PRINT"[illegible]"
1020 COLOR 1,3,6,0
1030 END
1050 DATA 3,6,"CANADA"
1060 DATA 20,6,"MEXICO"
1070 DATA 12,8,"USA"

READY.
```

RAINBOW

DESCRIPTION

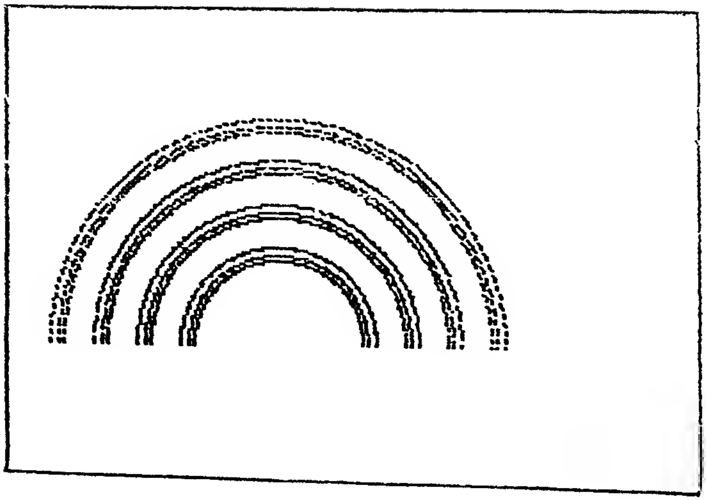
This program demonstrates how colours can be used with the high resolution plotting commands plus some of the limitations of high resolution colour. The display is a rainbow of four different coloured semicircles — red, yellow, green and blue. Each coloured semicircle is composed of three high resolution half circle plots. As the program stands the display produced has the four arcs each with a different colour, but notice that the gap between each arc is quite wide, try reducing the width of this gap and the colours of each arc start to break up. The gap can be reduced by changing the step value in line 210. The reason for this problem is simply that the colours are defined on a character square basis, trying to display two high resolution points of different colours in the same character space is impossible, the result is that the colour of the first plotted point will be changed to that of the second as soon as the second is plotted.

RUNNING THE PROGRAM

This program requires no input parameters, therefore simply enter RUN and watch the computer draw a coloured rainbow on the screen.

PROGRAM STRUCTURE

90	draw border around screen using subroutine at 500
110	coordinates of semicircle centre
120	start and end angle of semicircle
200-330	loop to draw four coloured arcs
410	colour data stored as colour values for each arc
500-560	border drawing subroutine



```
1 REM RAINBOW
2 REM ***********************
3 REM
10 REM THIS PROGRAM WILL DRAW A COLOURED RAINBOW
20 REM USING HIGH RESOLUTION
30 REM PLOTTING.
50 REM SET COLOURS
55 REM
60 GRAPHIC 2
70 COLOR 1,1,0,0
75 REM
80 REM DRAW BORDER
85 REM
90 GOSUB 500
95 REM
100 REM SET CONSTANTS
105 REM
110 XC=512:YC=900
120 P1=50:P2=100
130 REM
140 REM LOOP TO DRAW FOUR COLOURED RAINBOW
150 REM
160 FOR R=100 TO 650 STEP 150
170 READ C
180 REGION C
190 REM
200 REM THREE LINES TO EACH COLOUR
205 REM
210 FOR Q=20 TO 60 STEP 20
220 P=R+Q
230 CIRCLE 2,XC,YC,0.7*P,P,P1,P2
240 NEXT Q
250 NEXT R
300 REM END
310 GET A$:IF A$="" THEN 310
320 COLOR 1,3,6,0
330 GRAPHIC 0
340 END
395 REM
400 REM COLOUR DATA FOR RAINBOW
405 REM
410 DATA 2,7,5,6,
495 REM
500 REM DRAW BORDER
505 REM
510 POINT 2,0,0
520 DRAW 3 TO 0,1023
530 DRAW 3 TO 1023,1023
540 DRAW 3 TO 1023,0
550 DRAW 3 TO 0,0
560 RETURN
```

FAN

DESCRIPTION

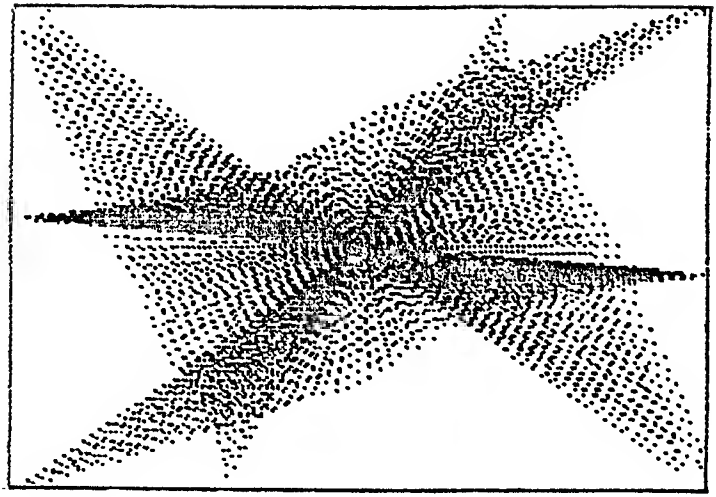
This is the last program in the section on colour and it simply produces a pretty changing and colourful pattern using high resolution colour plotting. The pattern is built up from different coloured high resolution lines and can be varied by changing the initial variable values in line 110 or by inserting extra loops into the main display loop — lines 140 to 220. The colour of each plotted line is set by a random value between 1 and 7 in lines 350-370. The lines are drawn by the subroutine 400 to 510.

RUNNING THE PROGRAM

Since no parameters are input by the program, simply type RUN and watch the pattern develop on the screen in constantly changing colours.

PROGRAM STRUCTURE

50	set background colour
110	initialisation variables — change for new pattern
140-220	main display loop — each of the four sub loops in this section draws a different part of the pattern, adds more sections or change values to change patterns
300-350	set values for line draw subroutine
360-370	set new line drawing colour
400-510	line drawing subroutine
600-660	border drawing subroutine



```
1 REM FAN
2 REM ***********************
3 REM
10 REM THIS PROGRAM DRAWS A
20 REM COLOURED ROTATING FAN
30 REM
40 REM SET BACKGROUND COLOUR
50 GRAPHIC 2
60 COLOR 1,1,0,0
75 REM
60 REM DRAW BORDER
85 REM
90 GOSUB 600
95 REM
100 REM SET UP VARIABLES
105 REM
110 X=0:Y=0:Q=25:Z=0
120 REGION 2
130 REM
135 REM MAIN LOOP
137 REM
140 FOR X=20 TO 1000 STEP 24
150 GOSUB 300:NEXT X
160 FOR Y=20 TO 1000 STEP 24
170 GOSUB 300:NEXT Y
180 FOR X=1000 TO 20 STEP -24
190 GOSUB 300:NEXT X
200 FOR Y=1000 TO 20 STEP -24
210 GOSUB 300:NEXT Y
220 GOTO 140
290 REM
300 REM DRAW LINE AND SET INK COLOUR
305 REM
310 XB=1000-X:YB=1000-Y
320 XE=X:YE=Y
330 GOSUB 400
340 Z=Z+1:IF Z>=Q THEN Z=0
350 Q=INT(RND(1)*50)
360 C=INT(RND(1)*7)+1
365 IF C=1 THEN 360
370 REGION C
380 RETURN
390 REM
400 REM DRAW LINE
410 A=XE-XB
420 B=YE-YB
430 Q=SQR(A*A+B*B)
440 UX=A/Q
450 UY=B/Q
460 FOR L=0 TO Q STEP 24
470 X1=XB+L*UX
```

```
480 Y1=YB+L*UY
490 IF X1<0 OR Y1<0 THEN 510
500 POINT 2,X1,Y1
510 NEXT L
520 RETURN
595 REM
600 REM DRAW BORDER
605 REM
610 POINT 2,0,0
620 DRAW 2 TO 0,1023
630 DRAW 2 TO 1023,1023
640 DRAW 2 TO 1023,0
650 DRAW 2 TO 0,0
660 RETURN

READY.
```

COLOURS

DESCRIPTION

Although the VIC has the ability to display a wide range of colours, and has some good high resolution graphics routines built into ROM, using both of these can at best be a little awkward. The use of the VIC cartridge the 'Super Expander' makes life considerably easier by giving you such commands as CIRCLE, PAINT, and so on. In this program we are going to use those two commands in particular to draw a circle, colour it in, and at the same time have you specify the colour of the 'paint' that we are going to use to fill the circle.

RUNNING THE PROGRAM

In this program only one input is required, and that is the colour of the paint to be used. Our input routine, commencing at line 100, but consisting mainly of the subroutine from lines 400 to 460, will only allow an input of a number from 0 to 9, or the '—' key. Inputting a negative number allows you to exit from the program, otherwise you just go back for another go. The central X and Y co-ordinates XC and YC (using the graphic 2 mode and a scaled resolution of 1024 by 1024) are set in lines 120 and 130 respectively, and the radius R is set in line 140. Our border drawing subroutine in lines 300 to 360 is used to DRAW a neat border around the screen, before drawing the circle and filling it in with the routine in lines 185 to 230. Line 240 then sends us back to request another colour.

PROGRAM STRUCTURE

60-70	set colours
90	draw border round screen using subroutine at 300
100	input colour using subroutine at 400
105	check for end of program
120	set X co-ordinate of centre of circle
130	set Y co-ordinate of centre of circle
140	set radius of circle
150	draw border round screen using subroutine at 300
185-240	draw circle and paint it in
250-280	end routine
300-360	border drawing subroutine
400-460	data input and checking routine

```
1 REM COLOURS
2 REM ***********************
3 REM
10 REM PROGRAM TO DRAW A CIRCLE
20 REM AND COLOUR IT IN USING THE
30 REM 'PAINT' COMMAND.COLOUR OF
40 REM PAINT IS INPUT.
45 REM
50 REM SET COLOURS
55 REM
60 GRAPHIC 2
70 COLOR 3,3,0,0
75 REM
80 REM DRAW BORDER
85 REM
90 GOSUB 300
95 REM
100 REM INPUT COLOUR OF PAINT
101 Z$="":T=5
102 CHAR 19,2,"?                    "
103 GOSUB 400
104 C=Z
105 IF C<0 THEN 250
106 IF C>7 THEN 100
107 FOR I=1 TO 500:NEXT I
110 REM SET OTHER PARAMETERS
115 REM
120 XC=512
130 YC=475
140 R=350
150 REM DRAW BORDER
155 REM
160 SCNCLR
170 GOSUB 300
180 REM
185 REM DRAW CIRCLE
190 REM
200 REGION C
205 REM
210 CIRCLE 2,XC,YC,0.7*R,R
220 PAINT 2,XC,YC
230 REGION 0
240 GOTO 100
250 REM END
260 COLOR 1,3,6,0
270 GRAPHIC 0
280 END
295 REM
300 REM DRAW BORDER
305 REM
310 POINT 2,0,0
```

```
320 DRAW 2 TO 0,950
330 DRAW 2 TO 1023,950
340 DRAW 2 TO 1023,0
350 DRAW 2 TO 0,0
360 RETURN
395 REM
400 REM INPUT DATA
405 REM
410 GET A$:IF A$="" THEN 410
420 IF (ASC(A$)<48 OR ASC(A$)>57) AND A$<>"-" THEN 450
430 CHAR 19,T,A$:T=T+1
440 Z$=Z$+A$:GOTO 410
450 Z=VAL(Z$)
460 RETURN

READY.
```

HIGH RESOLUTION

HIGH RESOLUTION DISPLAYS

Besides normal text the VIC can display drawings and shapes. Such graphics displays can be achieved using either the simple character graphics or the high resolution point plotting facility. Character graphics can be built up using strings of graphic characters displayed at the correct position on the screen. Such displays are, however, simple and crude; wherever possible, high resolution point plotting is preferable.

The easiest way to give the VIC high resolution point plotting capability is to use the Super Expander cartridge. This cartridge adds a range of useful high resolution point plotting commands to the VIC. If you do not possess this cartridge, a short program written in Basic, such as the example at the end of this section, can be used to plot high resolution points, lines etc.

THE THEORY OF HIGH RESOLUTION PLOTTING

The VIC has two display modes, normal text mode and user definable character mode. The modes are determined by the position in memory of the character generator. There are also two modes of colour operation, high resolution and multicolour. The VIC is thus capable of several permutations of colour and display mode.

The two display modes depend on whether the normal internal ROM-based character generator is used or a user definable RAM character generator. The position of the character generator within processor memory space is determined by the contents of bits 0-3 of control register # 5. These four bits form bits A10 to A13 of the actual character generator address as follows:

The normal contents of bits 0-3 of control register # 5 are zero; the way the VIC is configured, this gives a character generator address of Hex $8000 (decimal 32768). Starting at this location is a 4K ROM, the character generator; this contains the usual dot pattern for each of the 256 different characters which can be displayed. The 4K character generator ROM contains two separate character generators each occupying 2K of ROM.

The first of these two character generators which starts at address Hex $8000 (decimal 32768) contains the dot pattern for the 128 normal upper case and graphics characters plus the 128 reverse field versions of the same characters. The second character generator starts at location $8800 (decimal 34816) and is identical to the first except that part of the graphics character set is replaced by lower case characters. When the second character set is enabled the VIC will normally display in lower case characters rather than the normal upper case; upper case will be displayed with the shift key depressed.

The second character generator can be enabled normally by pressing the shift key and the Commodore logo key simultaneously. Alternatively, one can change the contents of control register # 5, thus:

POKE 36869,242: set lower case display mode
POKE 36869,240: set upper case display mode

This simply shifts the starting address of the character generator up 2K in memory, thereby accessing the second character generator.

The character generator starting address in control register # 5 can be changed so that the character generator is located in RAM, thereby allowing user definable characters to be created. The starting address of the user definable RAM character generator on the VIC can be any 2K (4K if 8 × 16 characters are used) block of RAM located between address Hex $1000 and $3000. It should be located at the highest possible address and protected from being overwritten by Basic by lowering the top of memory pointers to protect the RAM space used by the character generator. The setting up of control register # 5 has the following rules:

1) The starting address is always located at the beginning of a 1K block.
2) If the contents of bits 2 and 3 are both zero then the starting address defaults to the ROM at $8000 plus the offset stored in bits 0 and 1; this offset is in increments of 1K.
3) Bits 2 and 3 contain the starting address in increments of 4K.

Thus, to put the user definable character generator to start at 11K up in memory or Hex $2000 or 2 × 4K block plus 3 × 1K block, then bits 0 to 3 would be set up as follows:

Bits	3	2	1	0
Binary contents	1	0	1	1
Representing	2 × 4K blocks		3 × 1K blocks	

The user definable character generator is very important since it not only allows special graphics characters to be created but it also allows high resolution point plotting on the VIC. This allows a graph or display to be created with a resolution of 176 points in the horizontal by 184 points vertically, sufficient to give a very good quality display. High resolution point plotting is achieved by programming techniques using the user definable character generator. The use of the RAM character generator must be understood before these techniques can be explained.

The first stage in creating a user definable character set is to allocate a block of RAM memory for storage of the character generator. If characters on an 8 × 8 matrix are being displayed then 2048 memory locations are required; if an 8 × 16 matrix is to be used, then 4096 locations are required. Since a standard VIC only has 3584 RAM memory locations available to the user, an 8 × 8 matrix user definable character generator using 2048 of these locations is the only one feasible. The user RAM on a standard unexpanded VIC starts at memory address 4096 and goes on to address 7679.

The character generator can be programmed to start at any of the following addresses within that range: 4096, 5120, 6144 or 7168. Since 2048 locations are required for the character generator, the only possible starting location is 5120; this leaves 1024 bytes free for user programs (not much; purchase of the standard 3K RAM expansion module is strongly recommended; its use will not change the start address recommended above). This area of RAM chosen for use by the character generator must be protected from being overwritten by a Basic program or data; if this happened the display would be destroyed. The user definable character generator can be protected from being overwritten by lowering the top of memory pointers, thus:

```
10 POKE 51,255: POKE 52,19
11 POKE 55,255: POKE 56,19
12 CLR
```

The next stage is to put the data about each character into the new character generator. This is done by using POKE commands or machine code load statements to put information into the 2048 memory locations. Before this can be done each of the new characters must be designed; this entails drawing each character on an 8 × 8 grid (see Fig 1). Once the character has been designed it can be converted into the block of eight numerical values for storage in the character generator. Each line in the 8 × 8 grid corresponds to a byte of data and each of the eight bits in that byte corresponds to a dot or column position on that line.

Information is stored in memory in binary; thus by considering each bright dot to be a logical '1' and each space a logical '0', a line of dots in each character can be converted into a numerical value. The way this is done is shown in Fig 2. Some examples of character designs and their conversion to numerical values are shown in Fig 3. From these values a table can be created, one column having the character generator address and the corresponding entry in the second column having the value to be put into that location.

The table is divided into blocks of eight entries, each block containing the data for one character. Each of these blocks of eight entries is numbered starting at 0 and going up to 255. These numbers correspond to the ASCII or character code number stored in the video RAM when the characters are displayed. An example table using the character designs in Fig 3 is shown in Fig 4. The table need only contain the number of characters actually required; all 255 possible character blocks do not have to be filled in. It is advisable though that the table starts at the first location in the character generator; any gaps left should be filled with zeros. If the character generator is being loaded from a Basic program, the values in the table are best stored as DATA statements; these values are then entered into memory using POKE commands, thus:

```
20 FOR I=0 TO 2048
21 READ A
22 IF A="*" THEN 30
23 POKE 5120+I, A
24 NEXT
30 END

100 DATA 24,20,20,18,48,112,96,0
110 DATA 0,24,60,126,255,24,36,66
120 DATA 255,126,60,24,24,60,126,255
130 DATA *
```

In the majority of applications alphanumeric characters are required in addition to user defined graphics characters; in such cases part of the data in the ROM based character generator must be transferred to the new RAM character generator. All the alphanumeric characters plus the VIC graphics characters (or lower case depending on which of the two character generators is accessed) are contained in the first 128 characters of the character generator. The remaining 128 characters are the reverse field versions of the first 128 characters. The first 128 characters of the ROM character generator are transferred to the new RAM character generator using a combination of PEEK and POKE commands thus:

```
20 FOR I=0 TO 1024
30 POKE 5120+I, PEEK(32768+I)
40 NEXT I
```

This leaves 128 possible user definable characters starting at address 6155. These characters can be filled as described above, and will have an ASCII code starting value of 128. An example of the routine to enter the character generator data will be as follows:

```
20 FOR I=0 TO 1024
21 POKE 5120+I, PEEK(32768+I)
22 NEXT I

30 FOR I=0 TO 1024
31 READ A
32 IF A="*" THEN 200
33 POKE 6144+I, A
34 NEXT

60 REM DATA FOR ASCII CODE CHARACTERS 128, 129,
AND 130

100 DATA 24,20,20,18,48,112,96,0
110 DATA 0,24,60,126,255,24,36,66
120 DATA 255,126,60,24,24,60,126,255
130 DATA *
```

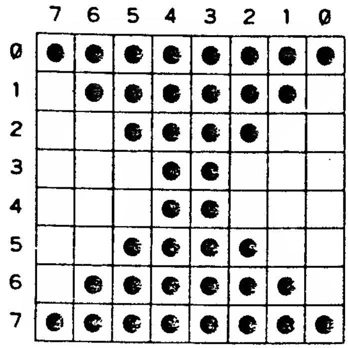
Having loaded the user definable character generator it can be used. It will remain in the VIC until the machine is switched off and can thus be used by more than one program. To use the RAM character generator two of the 6561 registers must be changed, thus:

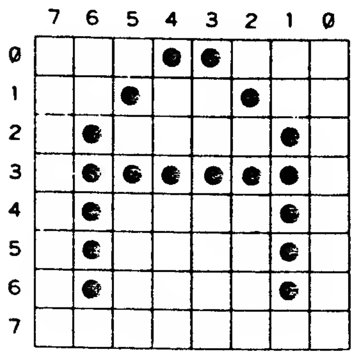
```
200 POKE 36869, 253
210 POKE 36866, PEEK(36866) OR 128
```

Once the user definable RAM character generator has been set up and the 6561 registers changed to utilise the new character generator, it can be used to generate special displays. If POKE commands are used to place the characters in the video RAM memory then the ASCII code value of the new characters is used. If the new characters are incorporated into strings then it is essential to know which character in the normal character set the new character replaces. This can be determined by using the table of VIC ASCII codes and looking for the character with the same code value as the new character. When the program is written the normal characters are inserted into the string; when the program is run they will be automatically replaced by the new characters. It is important to note that when using POKE commands the colour

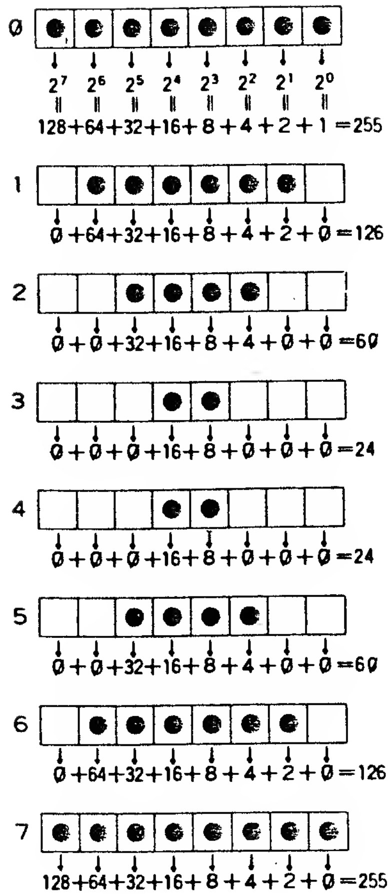
RAM location corresponding to the location where the character is to be displayed must also be set to give the required colour, otherwise the display will be white on white and therefore invisible. To restore the normal function of the VIC ROM character generator, use the following two lines:

```
500 POKE 36869,240
510 POKE 36866,150
```

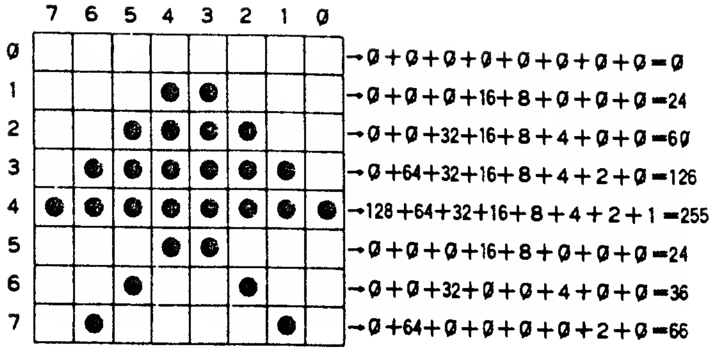


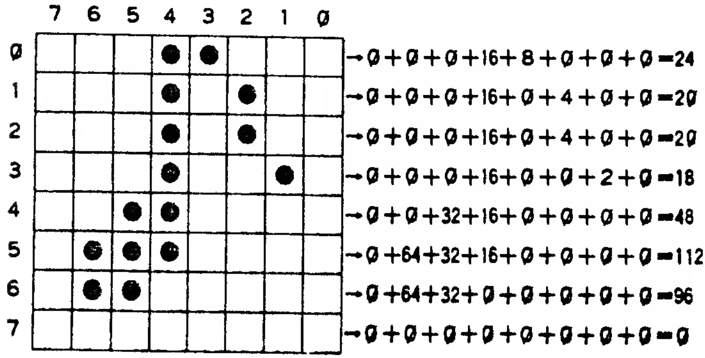


Examples of layout in design of characters.



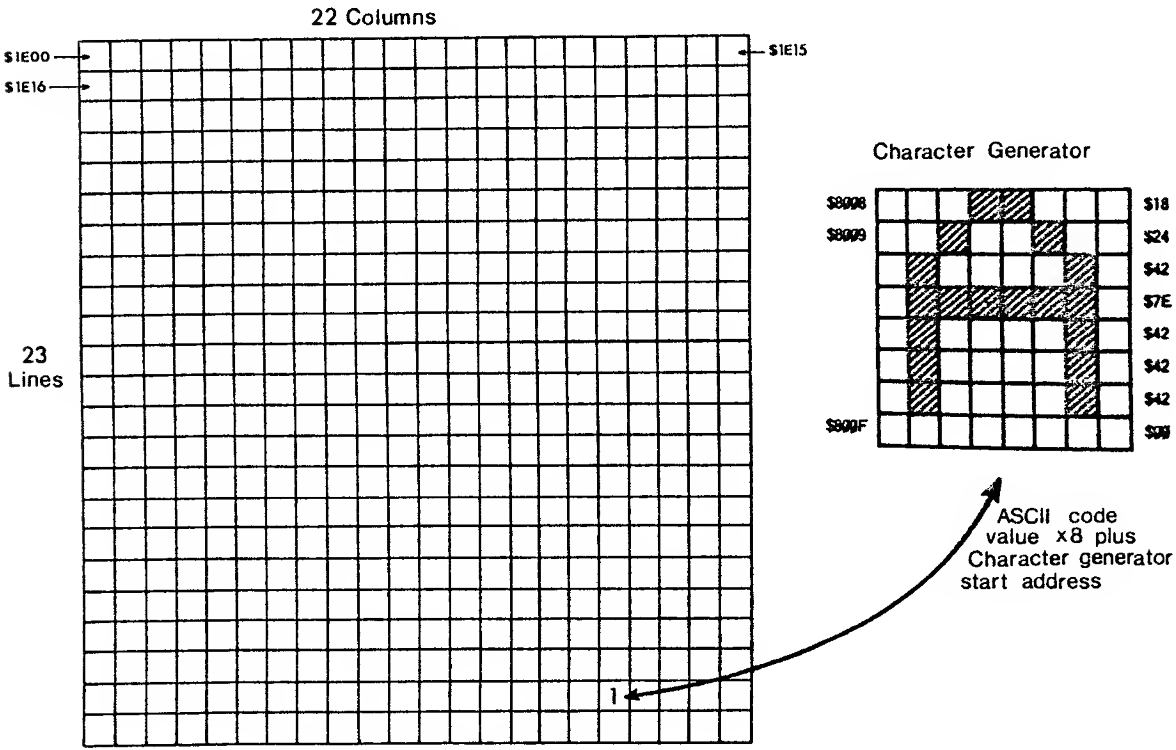
Conversion of a character into numerical values.





5120 – 0		
5121 – 24		
5122 – 60		
5123 – 126		
5124 – 255	Character # 1	
5125 – 24		
5126 – 36		
5127 – 66		

5128 – 24	
5129 – 20	
5130 – 20	Character # 2
5131 – 18	
5132 – 48	
etc.	



Video RAM mapped onto screen

High resolution point plotting uses exactly the same principles as the generation of user definable characters. it entails filling the video RAM with each of the 255 character codes (only half the screen can be used with 8 × 8 characters). The RAM character generator can then be used as a high resolution memory mapped display. If all bytes in the RAM character generator are set to zero then the screen is blank; set one bit in one of the characters and a single high resolution dot will appear on the screen.

The relationship between a single dot on the screen, the locations in the RAM character generator and the code value in each of the video memory locations is shown in Figure 6. This shows that the basis of high resolution plotting is simply filling the video RAM corresponding to the screen area of the high resolution display with successive and incremented code values. The rest is a matter of calculation to ensure that the correct bits are set in each of the eight bytes corresponding to each of the character codes used in the video RAM.

A high resolution plotting program consists of two parts, the initialisation and the point plot subroutine. The initialisation sets up the registers of the 6561 for the user definable character generator, lowers the top of memory to protect that character generator, puts the correct data into the video and colour RAMs and clears the contents of the RAM character generator. The point plot subroutine is called whenever a point is to be plotted or erased and consists of a routine which calculates, from given X and Y coordinates, which bit in which byte of the RAM character generator is to be set or erased.

It should be noted that the area of the screen devoted to high resolution plotting can vary from just a few adjacent character spaces to the whole screen (to do this the 6561 is initialised to display 8 × 16 characters rather than the normal 8 × 8; this requires the RAM character generator to be enlarged to 4K). An example of a set of Basic routines to plot points in high resolution, plus lines and circles, is contained in the following program (these routines use a 2K character generator and 8 × 8 characters so the display only occupies half the screen; the 6561 registers have been used to centre the display). Note also the routine which transfers characters from the ROM character generator to the user definable RAM character generator.

```
1 REM ****************************************
2 REM *PROGRAM TO PLOT THE GRAPH OF A FUNCTION
3 REM *IN HIGH RESOLUTION ON THE VIC
4 REM ****************************************
5 REM
6 REM * INITIALISE 6561 REGISTERS
7 PRINT"♥"
8 POKE36867,128
9 POKE36865,60
10 F(8)=0:F(0)=128:F(1)=64:F(2)=32:F(3)=16
20 F(4)=8:F(5)=4:F(6)=2:F(7)=1
30 FORQ=0TO255
32 POKE7680+Q,Q
34 POKE38400+Q,2
36 NEXTQ
40 FORQ=5120TO5120+255*8
42 POKEQ,0
44 NEXTQ
45 POKE36869,253
46 POKE36866,PEEK(36866)OR128
47 POKE36867,150
60 REM
61 REM *PLOT GRAPH OF FUNCTION IN LINE 90
62 REM
80 FORC=0TO175
90 L=45+40*SIN(C/10)
91 REM
92 REM *HIGH RESOLUTION POINT PLOT ROUTINE
93 REM
100 A=5120
110 LR=L/8
120 LA=INT(LR)
130 A=A+(LA*176)
140 LR=(LR-LA)*8
300 CR=C/8
310 CA=INT(CR)
320 A=A+(CA*8)
325 A=A+LR
330 CR=INT((CR-CA)*8)
400 POKEA,PEEK(A)ORF(CR)
500 NEXTC
550 REM
551 REM *WAIT FOR KEY PRESS THEN RETURN
552 REM *SCREEN TO NORMAL.
553 REM
600 GETA$:IFA$=""THEN600
1000 POKE36869,240
1010 POKE36866,150
1020 POKE36867,174
1030 POKE36865,38
```

```
1 REM ***************************************
2 REM *PROGRAM TO PLOT HIGH RESOLUTION
3 REM *POINTS, LINES AND CIRCLES ON THE VIC.
4 REM ***************************************
5 REM
6 REM *INITIALISE 6561 AND CHAR GEN
7 REM
8 POKE36867,128
9 POKE36865,60
10 F(8)=0:F(0)=128:F(1)=64:F(2)=32
20 F(3)=16:F(4)=8:F(5)=4:F(6)=2:F(7)=1
35 FORQ=0TO255
37 POKE7680+Q,Q
38 POKE38400+Q,2
39 NEXTQ
40 FORQ=5120TO5120+255*8
41 POKEQ,0
42 NEXTQ
45 POKE36869,253
46 POKE36866,PEEK(36866)OR128
47 POKE36867,150
90 REM
91 REM *DATA FOR LINE DRAWING
92 REM *START AT COORDINATES X1,Y1
93 REM *END AT COORDINATES X2,Y2
94 REM
100 READX1,Y1,X2,Y2
105 IFX1=255THEN200
110 GOSUB1000
120 GOTO100
150 DATA 80,10,100,40
151 DATA 80,10,60,40
152 DATA 95,38,95,80
153 DATA 65,38,65,80
154 DATA 65,80,95,80
155 DATA 85,80,85,60
156 DATA 90,80,90,60
157 DATA 85,60,90,60
158 DATA 70,75,70,60
159 DATA 75,75,75,60
160 DATA 70,75,75,75
161 DATA 70,60,75,60
162 DATA 70,50,70,35
163 DATA 75,50,75,35
164 DATA 70,50,75,50
165 DATA 70,35,75,35
166 DATA 85,50,85,35
167 DATA 90,50,90,35
168 DATA 85,50,90,50
169 DATA 85,35,90,35
170 DATA 20,80,20,50
```

```
171 DATA 22,80,22,50
172 DATA 120,80,120,50
173 DATA 122,80,122,50
188 REM *END OF LINE DATA
189 DATA 255,255,255,255
190 REM
191 REM *DATA FOR DRAWING CIRCLES
192 REM *CENTRE AT COORDINATES CX,CY
193 REM *RADIUS R
194 REM
199 DATA 255,255,255,255
200 CX=21:CY=40:R=10
210 GOSUB3000
220 CX=121:CY=35:R=15
230 GOSUB3000
240 GETA$:IFA$=""THEN240
1000 REM
1010 REM *LINE DRAWING ROUTINE
1020 REM *USES DATA FROM LINE DATA TABLE
1030 REM
1200 XD=X2-X1
1210 YD=Y2-Y1
1230 A0=1:A1=1
1240 IFYD<0THENA0=-1
1250 IFXD<0THENA1=-1
1270 XE=ABS(XD):YE=ABS(YD):D1=XE-YE
1280 IFD1>=0THEN1320
1290 S0=-1:S1=0:LG=YE:SH=XE
1300 IFYD>=0THENS0=1
1310 GOTO1340
1320 S0=0:S1=-1:LG=XE:SH=YE
1330 IFXD>=0THENS1=1
1340 REM
1350 TT=LG:TS=SH:UD=LG-SH:CR=LG-SH/2
1355 D=0
1360 REM
1370 C=X1:L=Y1:GOSUB2100
1380 IFCT>=0THEN1420
1390 CT=CT+TS:X1=X1+S1:Y1=Y1+S0
1410 GOTO1460
1420 CT=CT-UD:X1=X1+A1:Y1=Y1+A0
1460 TT=TT-1
1470 IFTT<0THENRETURN
1480 GOTO1370
2000 REM
2010 REM *POINT PLOT ROUTINE
2020 REM *USED BY LINE AND CIRCLE DRAW
2030 REM *ROUTINES
2040 REM *C=X COORDINATE
2050 REM *L=Y COORDINATE
2060 REM
```

```
2100 A=5120
2110 LR=L/8
2120 LA=INT(LR)
2130 A=A+(LA*176)
2140 LR=(LR-LA)*8
2300 CR=C/8
2310 CA=INT(CR)
2320 A=A+(CA*8)
2325 A=A+LR
2330 CR=INT((CR-CA)*8)
2400 POKEA,PEEK(A)ORF(CR)
2500 RETURN
2600 GETA$:IFA$=""THEN2600
3000 REM
3001 REM *CIRCLE DRAWING ROUTINE
3002 REM *OX AND OY ARE OFFSET VARIABLES
3003 REM *WHICH DETERMINE WHETHER A CIRCLE
3004 REM *OR ELIPSE IS DRAWN
3005 REM
3010 OX=1:OY=1.2
3020 A=2*π
3030 N=100
3040 INC=(A-0)/N
3050 FORI=0TOASTEPINC
3060 X=R*SIN(I):X=INT(X*OX+CX+.499)
3070 Y=R*COS(I):Y=INT(Y*OY+CY+.499)
3080 L=Y:C=X:GOSUB2100
3090 NEXTI
3100 RETURN
```

LINE

DESCRIPTION

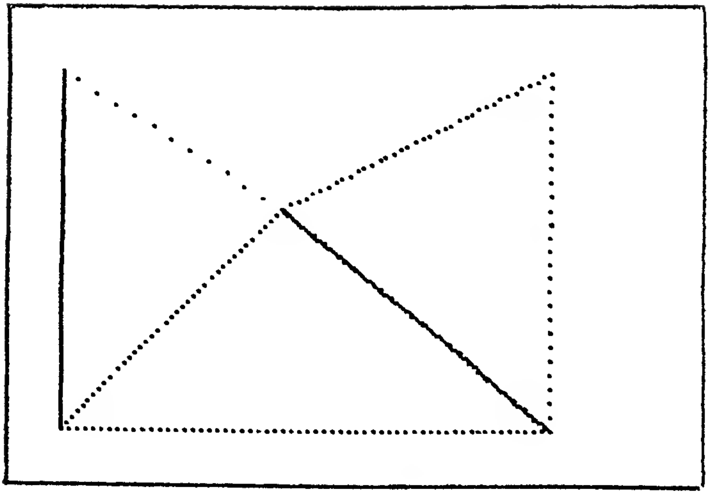
Although the VIC Super Expander command DRAW will draw a high resolution line between two points on the screen it has several serious drawbacks. Foremost of these drawbacks is that it uses relative coordinates, which are not very easy to use in many graphics applications. Another drawback is that it is impossible to draw a line with variable spacing between the dots. Both these problems are overcome by using this program, although it has one shortcoming in that since it is written in Basic it is rather slow. Most of the programs in this book which require line drawing use this routine. The variable R$ is input to determine if the line is to be drawn or erased (the line is erased if R$ = E).

RUNNING THE PROGRAM

In the program 'LINE' there are six variables which are input by program lines 100 to 130. The first two are input by line 100 and are the X, Y coordinates of the beginning of the line. The second two variables are the X and Y coordinates of the end of the line, and the last variable is the spacing between the dots used to draw the line. The input in line 120 determines whether the line is drawn or erased. If an 'E' is input then the line will be erased, if any other letter then the line will be drawn.

PROGRAM STRUCTURE

60-70	set colours
90	draw border around screen using subroutine at 500
100-150	input variables for start and end of line coordinates and dot spacing
160-320	line drawing routine
500-560	border drawing subroutine
600-660	Data input routine



```
1 REM LINE
2 REM ***********************
3 REM
10 REM THIS PROGRAM DRAWS OR ERASES A LINE
20 REM BETWEEN TWO SETS OF COORDINATES
30 REM THE SPACING BETWEEN THE DOTS USED
40 REM IS VARIABLE.
45 REM
50 REM SET COLOURS
55 REM
60 GRAPHIC 2
70 COLOR 3,3,0,3
75 REM
80 REM DRAW BORDER
85 REM
90 GOSUB 500
95 REM
100 REM LINE DRAWING ROUTINE PARAMETER INPUT
110 REM COORDINATES OF BEGINNING OF LINE
111 Z$="":T=5
112 CHAR 19,2,"?"
113 GOSUB 600
114 XB=Z:Z$=""
115 CHAR 19,T,",":T=T+1
116 GOSUB 600
117 YB=Z:FOR I=1 TO 500:NEXT I
118 IF XB<0 OR YB<0 THEN 700
119 CHAR 19,2,"                    "
120 REM COORDINATES OF END OF LINE
121 Z$="":T=5
122 CHAR 19,2,"?"
123 GOSUB 600
124 XE=Z:Z$=""
125 CHAR 19,T,",":T=T+1
126 GOSUB 600
127 YE=Z
128 FOR I=1 TO 500:NEXT I
129 CHAR 19,2,"                    "
130 REM DOT SPACING
131 Z$="":T=5
132 CHAR 19,2,"?"
133 GOSUB 600
134 DS=6*Z
135 FOR I=1 TO 500:NEXT I
136 CHAR 19,2,"                    "
140 REM DRAW OR ERASE
141 CHAR 19,2,"?"
142 GET R$:IF R$="" THEN 142
143 IF ASC(R$)<65 OR ASC(R$)>90 THEN 142
144 CHAR 19,5,R$
145 FOR I=1 TO 500:NEXT I
```

```
146 CHAR 19,2,"                    "
150 REM
160 REM DRAW LINE
170 REM
180 P=XE-XB
190 Q=YE-YB
200 R=SQR(P*P+Q*Q)
210 LX=P/R
220 LY=Q/R
230 FOR I=0 TO R STEP DS
240 X=XB+I*LX
250 Y=YB+I*LY
260 IF X>1023 OR Y>950 THEN 310
270 B=3
280 IF R$="E" THEN B=4
290 POINT B,X,Y
310 NEXT I
320 GOTO 100
495 REM
500 REM BORDER DRAWING ROUTINE
505 REM
510 POINT 3,0,0
520 DRAW 3 TO 0,950
530 DRAW 3 TO 1023,950
540 DRAW 3 TO 1023,0
550 DRAW 3 TO 0,0
560 RETURN
595 REM
600 REM INPUT DATA
605 REM
610 GET A$:IF A$="" THEN 610
620 IF (ASC(A$)<48 OR ASC(A$)>57) AND A$<>"-" THEN 650
630 CHAR 19,T,A$:T=T+1
640 Z$=Z$+A$:GOTO 610
650 Z=VAL(Z$)
660 RETURN
695 REM
700 REM END PROGRAM
705 REM
710 COLOR 1,3,6,0
720 GRAPHIC 0
730 END

READY.
```

RECTANGLE 1

DESCRIPTION

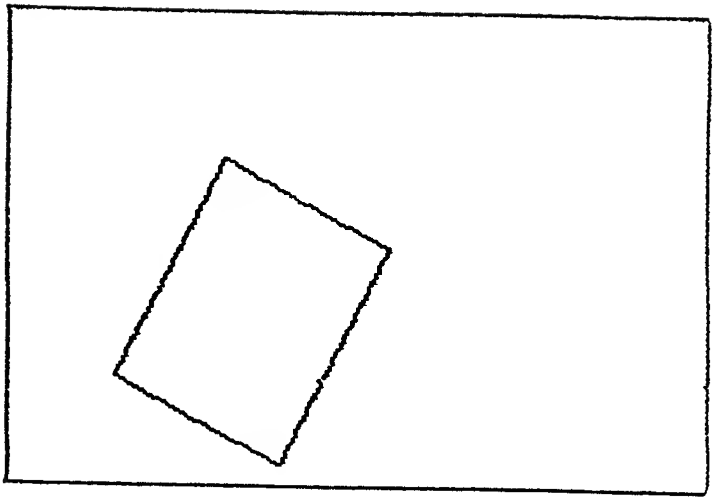
This program shows how to draw rectangles with sides which are not parallel to the screen axis. This is simply done by using a matrix of coordinates. Matrices are very important in graphics and an understanding of the principles is essential. The coordinate matrix is usually stored as data statements within the program and subsequently placed in an array. The values in this array can be manipulated mathematically, thereby allowing the shape to be rotated, scaled or moved about the screen area. All these will be dealt with in later sections of this book. In this program the values are simply used to display the shape at the specified coordinates.

RUNNING THE PROGRAM

Since all the coordinate values are stored in data statements — lines 210 and 220 — there are no values to be input in the program. However, to change the size or position of the rectangle it is necessary to input new data values into these data statements. Five coordinate values are required to draw the four lines of the rectangle; the X component of these five coordinates is stored in line 210 and the corresponding Y component in line 220. The best way to obtain these coordinate values for a new rectangle is to draw the shape with the correct scale and orientation onto graph paper and measure the required values.

PROGRAM STRUCTURE

50-60	set colours
80	draw border around screen using subroutine at 600
110-180	load matrix data into arrays
210	data for X component of coordinates
220	data for Y component of coordinates
300-350	set variables for draw
360	draw rectangle
600-660	border drawing subroutine



```
1 REM RECTANGLE
2 REM ***********************
3 REM
10 REM PROGRAM TO DRAW A RECTANGLE
20 REM USING MATRIX METHODS.
30 REM
40 REM SET COLOURS
50 GRAPHIC 2
60 COLOR 3,3,0,10
70 REM DRAW BORDER AROUND SCREEN
80 GOSUB 600
100 REM INPUT DATA FROM DATA STATEMENTS
110 REM INTO ARRAY.
120 DIM M(5,2)
130 FOR C=1 TO 5
140 READ M(C,1)
150 NEXT C
160 FOR C=1 TO 5
170 READ M(C,2)
180 NEXT C
200 REM DATA FOR COORDINATES
205 REM
210 DATA 160,320,560,400,160
220 DATA 800,400,560,960,800
300 REM DRAW RECTANGLE
310 FOR C=1 TO 4
320 XB=M(C,1)
330 YB=M(C,2)
340 XE=M(C+1,1)
350 YE=M(C+1,2)
360 DRAW 3,.7*XB,YB TO .7*XE,YE
365 NEXT C
370 GETA$:IFA$=""THEN370
380 COLOR1,3,6,0:GRAPHIC0:END
600 REM BORDER DRAWING ROUTINE
610 POINT 3,0,0
620 DRAW 3 TO 1023,0
630 DRAW 3 TO 1023,1023
640 DRAW 3 TO 0,1023
650 DRAW 3 TO 0,0
660 RETURN

READY.
```

POLYGON

DESCRIPTION

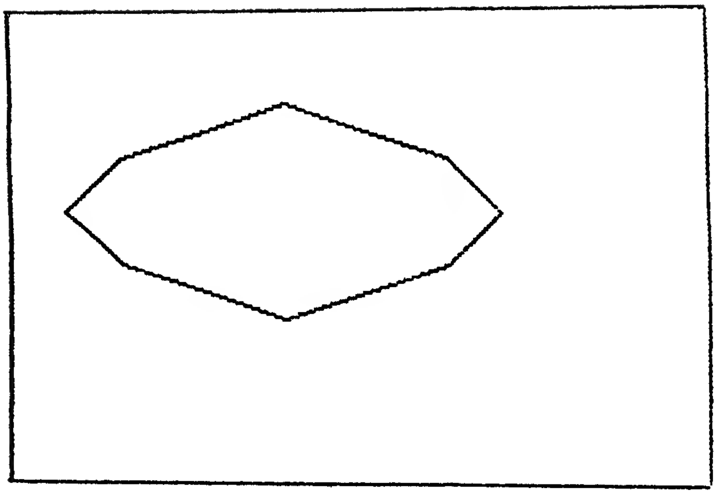
The only difference between this program and the previous program 'RECTANGLE' is the data used to draw the shape. The reason is that the use of a coordinate matrix is not confined to rectangles, it can be used to generate any required shape. In this program the data will draw an irregularly shaped octagon. To change the shape and its position simply change the data.

RUNNING THE PROGRAM

The coordinate data values are stored as data statements — lines 210 and 220 — so now there are no values to be input when the program is run. The size, shape or position of the shape on the screen can be changed by changing the data values in the data statements. It should be noted that when a shape is drawn the number of pairs of coordinate values is one more than the number of lines in the shape. The number of coordinate values used to draw the shape is stored as the first data statement value — line 205. The coordinates are stored as two sets of data, first all the X values and then in corresponding order all the Y values. In the example the X coordinates are thus stored in the data statement on line 210 and the Y values in line 220.

PROGRAM STRUCTURE

50-60	set colours
80	draw border around screen using subroutine at 600
110-180	load matrix data into arrays
205	number of coordinates in matrix data
210	data for X component of coordinates
220	data for Y component of coordinates
300-350	set variables for draw
360	draw polygon
600-660	border drawing subroutine



```
1 REM POLYGON
2 REM **************
3 REM
10 REM PROGRAM TO DRAW A POLYGON
15 REM WITH N SIDES USING MATRIX
20 REM METHODS.
40 REM SET COLOURS.
50 GRAPHIC2
60 COLOR3,3,0,10
70 REM DRAW BORDER
80 GOSUB 600
100 REM INPUT DATA FROM DATA STATEMENTS
110 REM INTO ARRAY.
115 READ N:REM NUMBER OF SIDES.
120 DIM M(N,2)
130 FOR C=1 TO N
140 READ M(C,1)
150 NEXT C
160 FOR C=1 TO N
170 READ M(C,2)
180 NEXT C
200 REM DATA FOR COORDINATE
205 DATA 9
210 DATA 100,200,512,824,924,824,512,200,100
220 DATA 500,400,300,400,500,600,700,600,500
300 REM DRAW POLYGON
310 FOR C=1 TO N-1
320 XB=M(C,1)
330 YB=M(C,2)
340 XE=M(C+1,1)
350 YE=M(C+1,2)
360 DRAW 3,XB,YB TO XE,YE
370 NEXT C
380 GET A$:IF A$="" THEN 380
390 COLOR 1,3,6,0:GRAPHIC 0:END
600 REM BORDER DRAWING ROUTINE
610 POINT 3,0,0
620 DRAW 3 TO 0,1023
630 DRAW 3 TO 1023,1023
640 DRAW 3 TO 1023,0
650 DRAW 3 TO 0,0
660 RETURN

READY.
```

POLYGON 2

DESCRIPTION

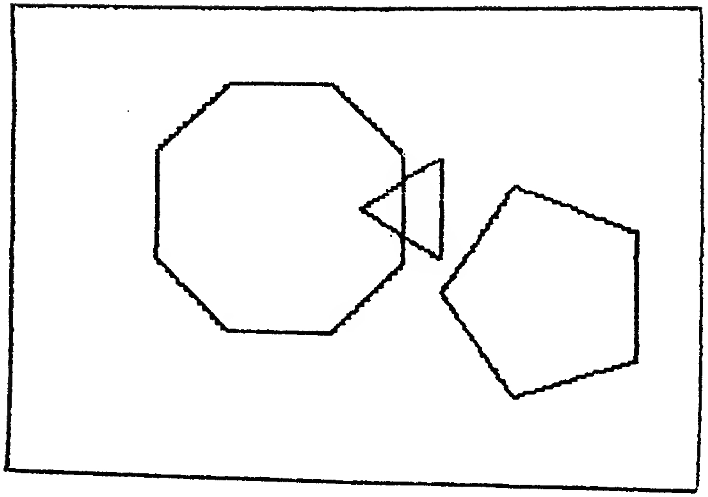
To save having to work out the end of line coordinates for each line of a polygon it is far easier given a regular N sided polygon to calculate these values within the program. This is done by the program POLYGON 2 which simply requires the centre of the polygon, the radius, the angular offset and the number of sides to the polygon. The program is configured to draw a series of polygons using data from a data table. The five parameters required to draw each polygon are then used to calculate a table of coordinates for each of the lines in the polygon, these values are then stored in the array m(n,2).

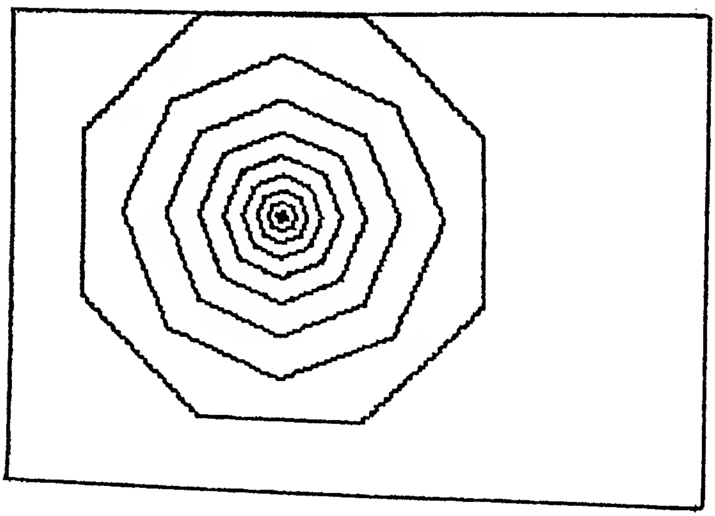
RUNNING THE PROGRAM

All the parameters required by the program are stored directly within the program. The X and Y coordinates of the central axis around which the shape is rotated is stored as the variables cx and cy. The number of lines in the shape is stored as variable n, r is the radius of the polygon and os is the angular offset. These values are stored as data statements in lines 300 to 320 (each line of datastatement holds the data for one polygon). To change the polygon's shape, orientation or position then change the values in the data statements, to add extra polygons then add further lines of data statement values.

PROGRAM STRUCTURE

60-70	set colours
90	draw border around screen using subroutine at 800
96	matrix for line coordinates and angles
140	get data from data statement for next polygon
180-190	convert angles to radians
200-220	calculate angles for each corner and put in array
300-320	data for drawing three polygons
400-460	calculate line coordinates and put in array
480-610	draw polygon
800-860	border drawing subroutine





```
1 REM POLYGON 2
2 REM **************
3 REM
10 REM PROGRAM TO DRAW N SIDED POLYGONS
20 REM GIVEN THE CENTRE,RADIUS,
30 REM AND ANGULAR OFFSET.
40 REM
50 REM SET COLOURS
60 GRAPHIC 2
70 COLOR 3,3,0,10
80 REM DRAW BORDER
90 GOSUB 800
95 Z=0
96 DIM M(10,2),A(10)
100 REM ROUTINE TO DRAW POLYGON
110 REM INPUT PARAMETERS FROM DATA
120 REM STATEMENTS AND SET UP MATRIX ARRAY.
130 REM
140 READ CX,CY,R,N,OS
145 Z=Z+1
150 Q=N+1
180 AD=2*π/N
190 OS=OS/180*π
200 FOR C=1 TO N
210 A(C)=C*AD+OS
220 NEXT C
300 DATA 400,400,200,8,22.5
310 DATA 600,400,80,3,60
320 DATA 800,720,160,5,36
395 REM
400 REM SET COORDINATES
405 REM
410 FOR X=1 TO N
420 M(X,1)=CX+R*COS(A(X))
430 M(X,2)=CY-R*SIN(A(X))
440 NEXT X
450 M(N+1,1)=M(1,1)
460 M(N+1,2)=M(1,2)
465 REM
470 REM DRAW POLYGON
475 REM
480 FOR C=1 TO N
490 XB=M(C,1)
500 YB=M(C,2)
510 XE=M(C+1,1)
520 YE=M(C+1,2)
530 DRAW 3,.7*XB,YB TO .7*XE,YE
540 NEXT C
550 IFZ<3THEN100
560 GET A$:IF A$="" THEN 560
570 COLOR 1,3,6,0:GRAPHIC0:END
610 DS=1:REM DOT SPACING
```

```
795 REM
800 REM BORDER DRAWING ROUTINE
810 POINT 3,0,0
820 DRAW 3 TO 0,1023
830 DRAW 3 TO 1023,1023
840 DRAW 3 TO 1023,0
850 DRAW 3 TO 0,0
860 RETURN

READY.
```

RECTANGLE 2

DESCRIPTION

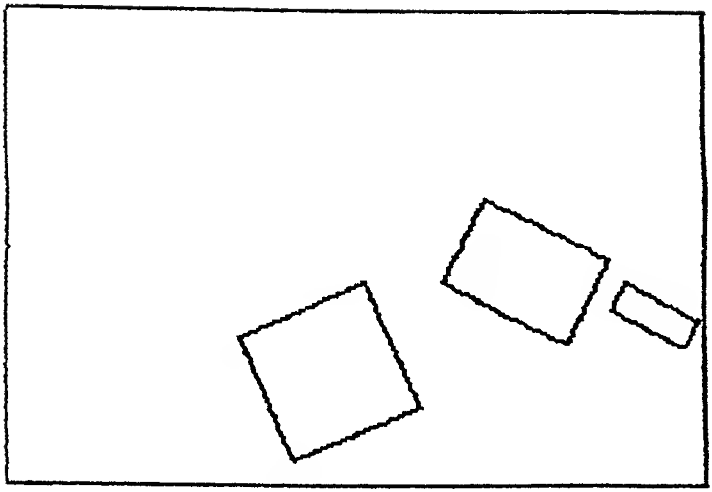
The problem with the program RECTANGLE 1 is that it requires the coordinates of all four corners. This program will draw rectangles of any orientation, given the coordinates of two corners and the length of one side, this is done using a simple calculation based on Pythagoras' Theorem to calculate a matrix of corner coordinates.

RUNNING THE PROGRAM

The program requires the input of five parameter values. The first two are the X and Y coordinates of the bottom left corner and next two values are the coordinates of the bottom right corner. The last value is the length of a side at right angles to the side described by the pair of coordinates points.

PROGRAM STRUCTURE

35	set up coordinate matrix array
50-60	set colours
80	draw border around screen using subroutine at 600
110-119	input bottom left X, Y coordinates
120-129	input bottom right X, Y coordinates
130-136	input length of perpendicular side
140-295	calculate all corner coordinates of the rectangle
300-400	draw rectangle
600-660	border drawing subroutine
700-760	input data subroutine
800-930	line drawing subroutine



```
1 REM RECTANGLE 2
2 REM **********************
3 REM
10 REM PROGRAM TO DRAW A RECTANGLE
20 REM GIVEN COORDINATES OF TWO CORNERS
30 REM AND LENGTH OF ONE SIDE.
35 DIM M(5,2)
40 REM SET COLOURS
50 GRAPHIC 2
60 COLOR 3,3,0,10
70 REM DRAW BORDER AROUND SCREEN
80 GOSUB 600
95 REM
100 REM INPUT DATA
110 REM INPUT X1,Y1
111 Z$="":T=5
112 CHAR 19,2,"?"
113 GOSUB 700
114 X1=Z:Z$=""
115 CHAR 19,T,",":T=T+1
116 GOSUB 700
117 Y1=Z:FOR I=1 TO 500:NEXT I
118 IF X1<0 OR Y1<0 THEN 390
119 CHAR 19,2,"                  "
120 REM INPUT X2,Y2
121 Z$="":T=5
122 CHAR 19,2,"?"
123 GOSUB 700
124 X2=Z:Z$=""
125 CHAR 19,T,",":T=T+1
126 GOSUB 700
127 Y2=Z
128 FOR I=1 TO 500:NEXT I
129 CHAR 19,2,"                  "
130 REM INPUT L
131 Z$="":T=5
132 CHAR 19,2,"?"
133 GOSUB 700
134 L=Z
135 FOR I=1 TO 500:NEXT I
136 CHAR 19,2,"                  "
140 P=X2-X1
150 Q=Y2-Y1
160 R=SQR(P*P+Q*Q)
170 LX=P/R
180 LY=Q/R
190 WX=-LY
200 WY=LX
210 M(1,1)=X1
220 M(2,1)=X2
230 M(3,1)=X2+WX*L
```

```
240 M(4,1)=X1+WX*L
250 M(5,1)=X1
260 M(1,2)=Y1
270 M(2,2)=Y2
280 M(3,2)=Y2+WY*L
290 M(4,2)=Y1+WY*L
295 M(5,2)=Y1
300 REM DRAW RECTANGLE
305 REM
310 FOR C=1 TO 4
320 XB=M(C,1)
330 YB=M(C,2)
340 XE=M(C+1,1)
350 YE=M(C+1,2)
360 GOSUB 800
370 NEXT C
380 GOTO 100
390 COLOR 11,3,6,0
400 GRAPHIC 0
410 END
600 REM BORDER DRAWING ROUTINE
605 REM
610 POINT 3,0,0
620 DRAW 3 TO 1023,0
630 DRAW 3 TO 1023,950
640 DRAW 3 TO 0,950
650 DRAW 3 TO 0,0
660 RETURN
695 REM
700 REM INPUT DATA
705 REM
710 GET A$:IF A$="" THEN 710
720 IF (ASC(A$)<48 OR ASC(A$)>57) AND A$<>"-" THEN 750
730 CHAR 19,T,A$:T=T+1
740 Z$=Z$+A$:GOTO 710
750 Z=VAL(Z$)
760 RETURN
795 REM
800 REM LINE DRAWING ROUTINE
805 REM
810 P=XE-XB
820 Q=YE-YB
830 R=SQR(P*P+Q*Q)
840 LX=P/R
850 LY=Q/R
860 FOR I=0 TO R STEP 6
870 X=.7*(XB+I*LX)
880 Y=YB+I*LY
890 IF X<0 OR Y<0 THEN 920
900 IF X>1023 OR Y>950 THEN 920
910 POINT 3,X,Y
```

```
920 NEXT I
930 RETURN

READY.
```

CIRCLE

DESCRIPTION

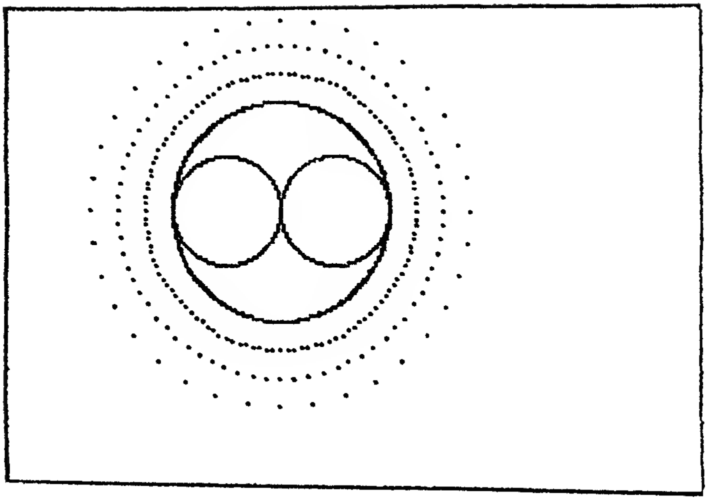
Plotting an ordinary circle with the VIC plus Super Expander is remarkably easy, using the built-in CIRCLE command, which allows you to specify the central X and Y coordinates, and also the radius. This will then plot a complete circle on the screen. However, for many applications we will not want a full circle, although we will require full image of the circle to be displayed. In other words, we want to be able to specify a distance between the points plotted that make up the circumference of the circle. The program CIRCLE does just that, by use of the POINT command to plot each individual dot of the circumference to a specified separation. This is the variable DS in the program listing, line 134.

RUNNING THE PROGRAM

A number of inputs are required to get the program going. In line 110 we input the X and Y coordinates of the centre of the circle, namely XC and YC, followed in line 120 by the radius RA. Our fourth input is the separation between the dots as mentioned earlier, that is the variable DS in line 130. This dot separation is then converted in line 210 (by multiplying by PI and dividing by 180) to form the STEP for the FOR NEXT loop in line 230 which initiates the plotting process. As we know, 2 PI radians equal 360 degrees, and hence the statement in line 230. Then we just calculate the distance of the dot in terms of X and Y coordinates from the centre of the circle, and POINT the point. Line 300 then sends us back for another run and another circle.

PROGRAM STRUCTURE

60-70	set colours
90	draw broder round screen using subroutine at 400
110	input coordinates of circle centre
120	input circle radius
130	input dot separation
210-330	draw circle
400-460	border drawing routine
500-560	input data subroutine



```
1 REM CIRCLE
2 REM ***********************
3 REM
10 REM ROUTINE TO DRAW A CIRCLE
20 REM SPACING BETWEEN THE DOTS USED
30 REM TO DRAW THE CIRCLE IS VARIABLE
40 REM
50 REM SET COLOURS
55 REM
60 GRAPHIC 2
70 COLOR 3,3,0,10
75 REM
80 REM DRAW BORDER
85 REM
90 GOSUB 400
95 REM
100 REM INPUT CIRCLE DRAWING PARAMETERS
110 REM COORDINATES OF CIRCLE CENTRE
111 Z$="":T=5
112 CHAR 19,2,"?"
113 GOSUB 500
114 XC=Z:Z$=""
115 CHAR 19,T,",":T=T+1
116 GOSUB 500
117 YC=Z:FOR I=1 TO 500:NEXT I
118 IF XC<0 OR YC<0 THEN 310
119 CHAR 19,2,"                  "
120 REM CIRCLE RADIUS
121 Z$="":T=5
122 CHAR 19,2,"?"
123 GOSUB 500
124 RA=Z
125 FOR I=1 TO 500:NEXT I
126 CHAR 19,2,"                  "
130 REM DOT SPACING
131 Z$="":T=5
132 CHAR 19,2,"?"
133 GOSUB 500
134 DS=Z
135 FOR I=1 TO 500:NEXT I
136 CHAR 19,2,"                  "
195 REM
200 REM DRAW CIRCLE
205 REM
210 DS=DS*π/180
220 R=RA
230 FOR P=0 TO 2*π STEP DS
240 X=R*COS(P)
250 Y=R*SIN(P)
260 X=.7*X+XC
270 Y=Y+YC
```

```
275 IF X<0 OR Y<0 OR Y>950 OR X>1023 THEN 290
280 POINT 3,X,Y
290 NEXT P
300 GOTO 100
310 COLOR1,3,6,0
320 GRAPHIC0
330 END
395 REM
400 REM BORDER DRAWING ROUTINE
405 REM
410 POINT 3,0,0
420 DRAW 3 TO 1023,0
430 DRAW 3 TO 1023,950
440 DRAW 3 TO 0,950
450 DRAW 3 TO 0,0
460 RETURN
495 REM
500 REM INPUT DATA
505 REM
510 GET A$:IF A$="" THEN 510
520 IF (ASC(A$)<48 OR ASC(A$)>57) AND A$<>"-" THEN 550
530 CHAR 19,T,A$:T=T+1
540 Z$=Z$+A$:GOTO 510
550 Z=VAL(Z$)
560 RETURN

READY.
```

ELLIPSE

DESCRIPTION

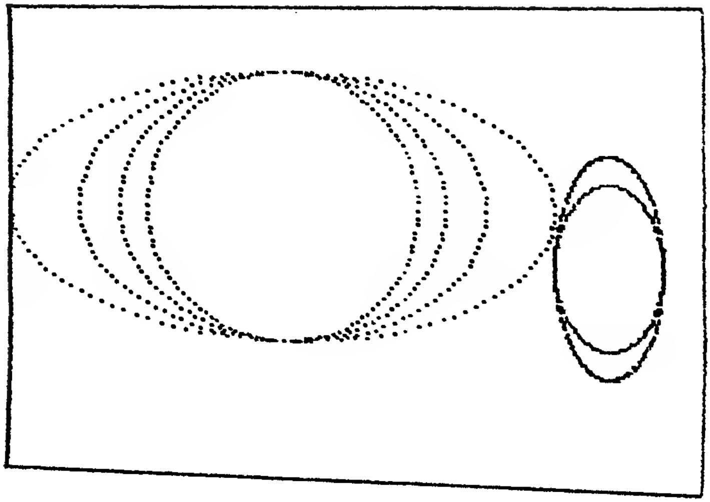
An ellipse is a circle offset on two sides from the central point in either the X or the Y direction. Using the routine developed in the program Circle, together with a couple of additions to handle the elliptical effect, we can plot an ellipse, or indeed any number of ellipses, with variable dot spacing. The offsets are specified in line 140, and determine the degree of ellipse. The variables OX and OY are used, and obviously if OX is zero we get an ellipse in the Y direction, and vice versa. Naturally we can give values to both of these to get a number of interesting effects.

RUNNING THE PROGRAM

In structure this is very similar to the Circle program earlier, but a couple of major differences are worthy of note. In line 140 we are asked to input the variables OX and OY to specify the degree of ellipse. These are subsequently used in our ellipse drawing routine in lines 240-250 to calculate precisely where our point is to be plotted. The rest of the program, including the routine to specify the separation of the dots (lines 210 and 230) is virtually the same.

PROGRAM STRUCTURE

60-70	set colours
90	draw border round screen using subroutine at 400
110	input coordinates of ellipse centre
120	input ellipse radius
130	input dot separation
140	input elliptical offsets in X and Y direction
210-360	draw ellipse
400-460	border drawing routine
500-560	data input subroutine



```
1 REM ELLIPSE
2 REM **********************
3 REM
10 REM ROUTINE TO DRAW AN ELLIPSE USING OFFSETS
20 REM SPACING BETWEEN THE DOTS USED
30 REM TO DRAW THE ELIPSE IS VARIABLE
40 REM
50 REM SET COLOURS
55 REM
60 GRAPHIC 2
70 COLOR 3,3,0,10
75 REM
80 REM DRAW BORDER AROUND SCREEN
85 REM
90 GOSUB 400
95 REM
100 REM INPUT ELLIPSE DRAWING PARAMETERS
105 REM
110 REM COORDINATES OF ELLIPSE CENTRE
111 Z$="":T=5
112 CHAR 19,2,"?"
113 GOSUB 500
114 XC=Z:Z$=""
115 CHAR 19,T,",":T=T+1
116 GOSUB 500
117 YC=Z:FOR I=1 TO 500:NEXT I
118 IF YC=0 OR XC=0 THEN 350
119 CHAR 19,2,"                    "
120 REM ELLIPSE RADIUS
121 Z$="":T=5
122 CHAR 19,2,"?"
123 GOSUB 500
124 RA=Z
125 FOR I=1 TO 500:NEXT I
126 CHAR 19,2,"                    "
130 REM DOT SPACING
131 Z$="":T=5
132 CHAR 19,2,"?"
133 GOSUB 500
134 DS=Z
135 FOR I=1 TO 500:NEXT I
136 CHAR 19,2,"                    "
140 REM ELLIPTICAL OFFSETS IN X AND Y AXIS
141 Z$="":T=5
142 CHAR 19,2,"?"
143 GOSUB 500
144 OX=Z:Z$=""
145 CHAR 19,T,",":T=T+1
146 GOSUB 500
147 OY=Z
148 FOR I=1 TO 500:NEXT I
```

```
149 CHAR 19,2,"                    "
195 REM
200 REM DRAW ELLIPSE
205 REM
210 DS=DS*π/180
220 R=RA
230 FOR P=0 TO 2*π STEP DS
235 REM
240 X=R*COS(P)*OX
250 Y=R*SIN(P)*OY
260 X=X*.7+XC
270 Y=Y+YC
275 IF X<0 OR Y<0 OR Y>950 THEN 290
280 POINT 3,X,Y
290 NEXT P
300 GOTO 100
350 COLOR 1,3,6,0
360 GRAPHIC 0
370 END
395 REM
400 REM BORDER DRAWING ROUTINE
405 REM
410 POINT 3,0,0
420 DRAW 3 TO 1023,0
430 DRAW 3 TO 1023,950
440 DRAW 3 TO 0,950
450 DRAW 3 TO 0,0
460 RETURN
495 REM
500 REM INPUT DATA
505 REM
510 GET A$:IF A$="" THEN 510
520 IF (ASC(A$)<48 OR ASC(A$)>57) AND A$<>"." THEN 550
530 CHAR 19,T,A$:T=T+1
540 Z$=Z$+A$:GOTO 510
550 Z=VAL(Z$)
560 RETURN

READY.
```

ARC 1

DESCRIPTION

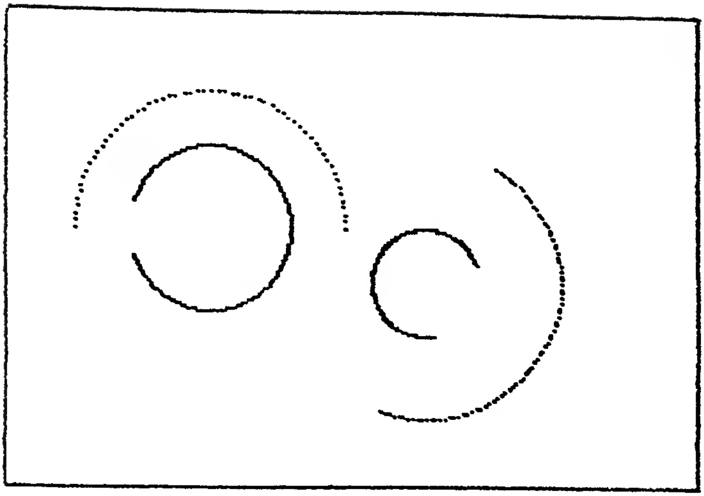
The VIC Super Expander command DRAW, while not being without its uses, suffers from a number of limitations. Like the CIRCLE command, you can only draw complete, filled in lines. Also, whether we use it in conjunction with the third parameter (other than X and Y coordinates of the finishing point), namely the angle through which it must turn, or not, we must always remember that DRAW will start off from the last point plotted by CIRCLE, POINT or the previous DRAW statement. In order to draw an arc from anywhere to anywhere, and to be able to have user-definable dot spacing, the routines in the program ARC1 were developed.

RUNNING THE PROGRAM

A number of inputs are required. In line 110 we must specify XC and YC, that is, the centre of the arc. Line 120 allows us to specify RA, the arc radius, and line 130 lets us input the dot separation DS. Two further inputs in line 140 contain the crux of the matter, and give us that much needed flexibility over DRAW, by allowing us to specify the start and end angles of the arc. Thus, we are not limited in where we can start drawing. The drawing routine in lines 250 to 310 is similar to the ones in earlier programs in this series.

PROGRAM STRUCTURE

60-70	set colours
90	draw border round screen using subroutine at 400
110	input coordinates of arc centre
120	input arc radius
130	input dot separation
140	input start and end angles for arc
210-360	draw arc
400-460	border drawing routine
500-560	data input subroutine



```
1 REM ARC 1
2 REM ***********************
3 REM
10 REM ROUTINE TO DRAW AN ARC
20 REM SPACING BETWEEN THE DOTS USED
30 REM TO DRAW THE ARC IS VARIABLE
40 REM
50 REM SET COLOURS
55 REM
60 GRAPHIC 2
70 COLOR 3,3,0,10
75 REM
80 REM DRAW BORDER AROUND SCREEN
85 REM
90 GOSUB 400
95 REM
100 REM INPUT ARC DRAWING PARAMETERS
105 REM
110 REM COORDINATES OF CENTRE OF ARC
111 Z$="":T=5
112 CHAR 19,2,"?"
113 GOSUB 500
114 XC=Z:Z$=""
115 CHAR 19,T,",":T=T+1
116 GOSUB 500
117 YC=Z:FOR I=1 TO 500:NEXT I
118 IF XC=0 OR YC=0 THEN 350
119 CHAR 19,2,"                    "
120 REM ARC RADIUS
121 Z$="":T=5
122 CHAR 19,2,"?"
123 GOSUB 500
124 RA=Z
125 FOR I=1 TO 500:NEXT I
126 CHAR 19,2,"                    "
130 REM DOT SPACING
131 Z$="":T=5
132 CHAR 19,2,"?"
133 GOSUB 500
134 DS=Z
135 FOR I=1 TO 500:NEXT I
136 CHAR 19,2,"                    "
140 REM START AND END ANGLES FOR ARC
141 Z$="":T=5
142 CHAR 19,2,"?"
143 GOSUB 500
144 AS=Z:Z$=""
145 CHAR 19,T,",":T=T+1
146 GOSUB 500
147 AE=Z
148 FOR I=1 TO 500:NEXT I
```

```
149 CHAR 19,2,"                              "
195 REM
200 REM DRAW ARC
205 REM
210 DS=DS*π/180
220 AS=AS*π/180
230 AE=AE*π/180
240 R=RA
250 FOR P=AS TO AE STEP DS
260 X=R*COS(P)
270 Y=R*SIN(P)
280 X=.7*X+XC
290 Y=Y+YC
295 IF X<0 OR Y<0 OR Y>950 THEN 310
300 POINT 3,X,Y
310 NEXT P
320 GOTO 100
350 COLOR 1,3,6,0
360 GRAPHIC 0
370 END
395 REM
400 REM BORDER DRAWING ROUTINE
405 REM
410 POINT 3,0,0
420 DRAW 3 TO 0,950
430 DRAW 3 TO 1023,950
440 DRAW 3 TO 1023,0
450 DRAW 3 TO 0,0
460 RETURN
495 REM
500 REM INPUT DATA
505 REM
510 GET A$:IF A$="" THEN 510
520 IF (ASC(A$)<48 OR ASC(A$)>57) AND A$<>"." THEN 550
530 CHAR 19,T,A$:T=T+1
540 Z$=Z$+A$:GOTO 510
550 Z=VAL(Z$)
560 RETURN

READY.
```

DISK 1

DESCRIPTION

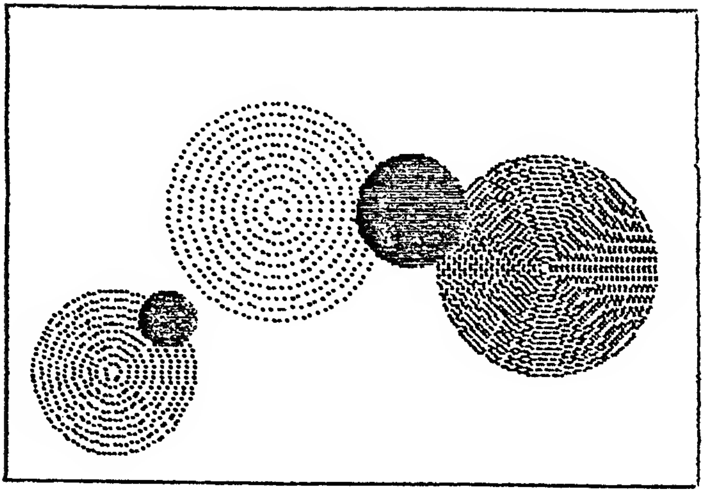
When examining the program CIRCLE, you probably realised that if you repeated the process again and again, but specifying a different radius each time, it would be possible to build up a complete disk rather than just a circle. This is certainly true, but the time taken would be rather a long one, and you'd probably get fed up with running through the program time after time. Consequently, the program DISK 1 takes the drudgery out of the process by incorporating a couple of new routines to do it all for you.

RUNNING THE PROGRAM

Again, we have to input a number of variables before we get to the meat of the program. As before, line 110 allows us to specify the coordinates of the disk centre, line 120 the disk radius, and line 130 the dot spacing. In drawing the disk however, we go through two FOR NEXT loops rather than the usual one. The inner loop, lines 230 to 290, draws just one circle as we've seen before. The loop in line 220 and 300 then uses the previously specified dot separation to step up the radius of the circle to draw another one, until finally we reach the full radius originally input in line 120.

PROGRAM STRUCTURE

60-70	set colours
90	draw border round screen using subroutine at 400
110	input coordinates of disk centre
120	input disk radius
130	input dot separation
210-290	draw arc, incorporating:-
230-290	draw circle, and
220-370	step up radius and draw another one
400-460	border drawing routine
500-560	data input subroutine



```
1 REM DISK 1
2 REM **********************
3 REM
10 REM ROUTINE TO DRAW A DISK
20 REM SPACING BETWEEN THE DOTS USED
30 REM TO DRAW THE DISK IS VARIABLE
40 REM
50 REM SET COLOURS
55 REM
60 GRAPHIC 2
70 COLOR 3,3,0,10
75 REM
80 REM DRAW BORDER AROUND SCREEN
65 REM
90 GOSUB 400
95 REM
100 REM INPUT DISK DRAWING PARAMETERS
105 REM
110 REM COORDINATES OF CENTRE OF DISK
111 Z$="":T=5
112 CHAR 19,2,"?"
113 GOSUB 500
114 XC=Z:Z$=""
115 CHAR 19,T,",":T=T+1
116 GOSUB 500
117 YC=Z:FOR I=1 TO 500:NEXT I
118 IF XC=0 OR YC=0 THEN 350
119 CHAR 19,2,"                    "
120 REM DISK RADIUS
121 Z$="":T=5
122 CHAR 19,2,"?"
123 GOSUB 500
124 RA=Z
125 FOR I=1 TO 500:NEXT I
126 CHAR 19,2,"                    "
130 REM DOT SPACING
131 Z$="":T=5
132 CHAR 19,2,"?"
133 GOSUB 500
134 DS=Z
135 FOR I=1 TO 500:NEXT I
136 CHAR 19,2,"                    "
195 REM
200 REM DRAW DISK
205 REM
210 D=DS*π/180
220 FOR R=DS TO RA STEP DS
230 FOR P=0 TO 2*π STEP D*(40/R)
240 X=R*COS(P)
250 Y=R*SIN(P)
```

```
260 X=.7*X+XC
270 Y=Y+YC
280 IF X<0 OR Y<0 OR Y>950 THEN 300
290 POINT 3,X,Y
300 NEXT P
310 NEXT R
320 GOTO 100
350 COLOR 1,3,6,0
360 GRAPHIC 0
370 END
395 REM
400 REM BORDER DRAWING ROUTINE
405 REM
410 POINT 3,0,0
420 DRAW 3 TO 0,950
430 DRAW 3 TO 1023,950
440 DRAW 3 TO 1023,0
450 DRAW 3 TO 0,0
460 RETURN
495 REM
500 REM INPUT DATA
505 REM
510 GET A$:IF A$="" THEN 510
520 IF (ASC(A$)<48 OR ASC(A$)>57) AND A$<>"." THEN 55
530 CHAR 19,T,A$:T=T+1
540 Z$=Z$+A$:GOTO 510
550 Z=VAL(Z$)
560 RETURN

READY.
```

SEGMENT

DESCRIPTION

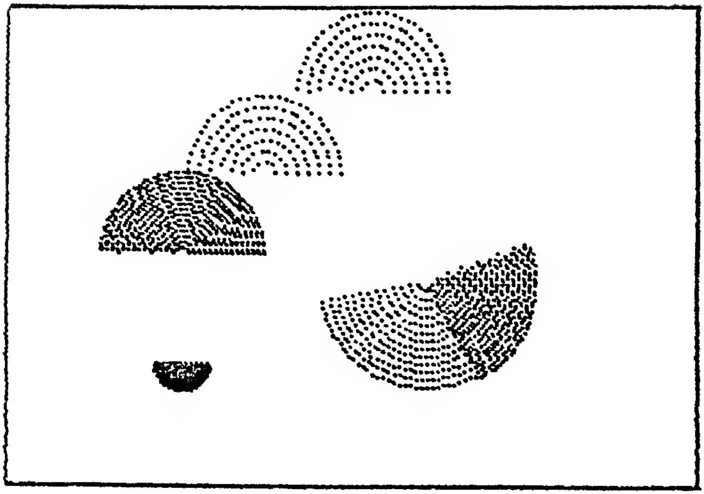
Although DRAW allows one to draw an arc, it does not allow one to draw an arc with variable dot spacing. By drawing various circles to variable dot spacing, a disk with the same dot spacing can be plotted. Combining both of these routines resulted in the program Segment, presented here. Using this program we can draw a disk segment, again with the spacing between the dots defined by an input (line 130), and moreover we can make that segment as large, or as small, as we like. As you can see from the illustration, combining a number of runs of the program enables us to link different disk segments together.

RUNNING THE PROGRAM

As usual, line 110 lets us input the coordinates of the arc centre, 120 the arc radius, and 130 the spacing between the dots. In line 140 we input the start and end angles for the arc. The program following is then fairly straightforward. In lines 250 to 310 we plot just one arc, using the POINT command for each point of the arc. The outer FOR NEXT loop, in lines 240 and 320, uses the dot separation to increase the radius of the arc, and then the inner loop plots out another arc. This continues until we reach the final radius of the arc, RA, as input in line 120, which gives us our final arc and completes the segment. By specifying a different dot spacing we can build up a whole series of arcs joined onto each other.

PROGRAM STRUCTURE

60-70	set colours
90	draw border using routine at 400
110	input central coordinates of arc
120	input radius of arc
130	input dot spacing
140	input start and end angles for arc
240,320	outer drawing routine, incorporating:
250-370	individual arc drawing routine
400-460	border drawing subroutine
500-560	data input subroutine



```
1 REM SEGMENT
2 REM ***********************
3 REM
10 REM ROUTINE TO DRAW A DISK SEGMENT
20 REM SPACING BETWEEN THE DOTS USED
30 REM TO DRAW THE SEGMENT IS VARIABLE
40 REM
50 REM SET COLOURS
60 GRAPHIC 2
70 COLOR 3,3,0,10
75 REM
80 REM DRAW BORDER AROUND SCREEN
85 REM
90 GOSUB 400
95 REM
100 REM INPUT SEGMENT DRAWING PARAMETERS
105 REM
110 REM COORDINATES OF SEGMENT CENTRE
111 Z$="":T=5
112 CHAR 19,2,"?"
113 GOSUB 500
114 XC=Z:Z$=""
115 CHAR 19,T,",":T=T+1
116 GOSUB 500
117 YC=Z:FOR I=1 TO 500:NEXT I
118 IF XC=0 OR YC=0 THEN 350
119 CHAR 19,2,"                    "
120 REM SEGMENT RADIUS
121 Z$="":T=5
122 CHAR 19,2,"?"
123 GOSUB 500
124 RA=Z
125 FOR I=1 TO 500:NEXT I
126 CHAR 19,2,"                    "
130 REM DOT SPACING
131 Z$="":T=5
132 CHAR 19,2,"?"
133 GOSUB 500
134 DS=Z
135 FOR I=1 TO 500:NEXT I
136 CHAR 19,2,"                    "
140 REM START AND END ANGLES FOR SEGMENT
141 Z$="":T=5
142 CHAR 19,2,"?"
143 GOSUB 500
144 AS=Z:Z$=""
145 CHAR 19,T,",":T=T+1
146 GOSUB 500
147 AE=Z
148 FOR I=1 TO 500:NEXT I
149 CHAR 19,2,"                    "
195 REM
```

```
200 REM DRAW SEGMENT
205 REM
210 D=DS*π/180
220 AS=AS*π/180
230 AE=AE*π/180
240 FOR R=DS TO RA STEP DS
250 FOR P=AS TO AE STEP D*(40/R)
260 X=R*COS(P)
270 Y=R*SIN(P)
280 X=.7*X+XC
290 Y=Y+YC
295 IF X<0 OR Y<0 OR Y>950 THEN 310
300 POINT 3,X,Y
310 NEXT P
320 NEXT R
330 GOTO 100
350 COLOR 1,3,6,0
360 GRAPHIC 0
370 END
395 REM
400 REM BORDER DRAWING ROUTINE
405 REM
410 POINT 3,0,0
420 DRAW 3 TO 0,950
430 DRAW 3 TO 1023,950
440 DRAW 3 TO 1023,0
450 DRAW 3 TO 0,0
460 RETURN
495 REM
500 REM INPUT DATA
505 REM
510 GET A$:IF A$="" THEN 510
520 IF (ASC(A$)<48 OR ASC(A$)>57) AND A$<>"." THEN 5
530 CHAR 19,T,A$:T=T+1
540 Z$=Z$+A$:GOTO 510
550 Z=VAL(Z$)
560 RETURN

READY.
```

PIECHART

DESCRIPTION

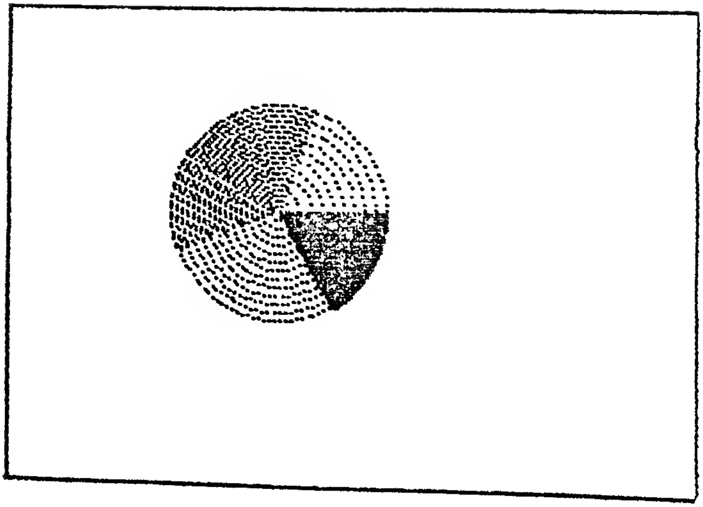
The culmination of all the plotting routines for circles, arcs, and disks results in the program PIECHART. Of use in business, educational, and indeed just about any computing environment, piecharts enable us to show clearly and (quite strikingly) visually all manner of different data. We mentioned when describing the program Segment, that by building up various runs through the program it was possible to have different segments next to each other. This program takes the chore out of that exercise, by assigning various variables first of all, and then using DATA statements to generate the necessary information. Obviously, this program will be of most use to you when using your own data.

RUNNING THE PROGRAM

This program differs from the earlier Segment one by having no input statements. Instead, we define the variables XC and YC to be the central coordinates in line 110, and the variable RA to be the radius in line 120. Needless to say you can change these to suit your own requirements. The data for making up the different arc segments is contained in lines 500 to 560. In order, we have the dot separation, the start angle for the segment, and the end angle. Again, these can be whatever you require. By reading these in lines 130 and 140, we then follow the segment plotting routine in lines 240 to 320. When line 150 detects a zero dot separation (as read in from line 540) the program comes to a halt.

PROGRAM STRUCTURE

60-70	set colours
90	draw border using routine at 400
110	define coordinates of centre of piechart
120	define radius of piechart
130	READ dot spacing
140	READ start and end angles of segment
150	if spacing of zero, then STOP
210-320	segment drawing routine
330	back for more data
400-460	border drawing subroutine
500-540	data for piechart



```
1 REM PIECHART
2 REM *********************
3 REM
10 REM ROUTINE TO DRAW A PIECHART USING
20 REM VARIABLE SPACING BETWEEN THE DOTS
30 REM TO DIFFERENTIATE BETWEEN SEGMENTS.
40 REM
50 REM SET COLOURS
60 GRAPHIC 2
70 COLOR 3,3,0,10
75 REM
80 REM DRAW BORDER
90 GOSUB 400
95 REM
100 REM GET PIECHART DATA FROM DATA TABLES - LINE 500+
105 REM
110 XC=512:YC=512:REM COORDINATES OF DISK CENTRE
120 RA=300:REM DISK RADIUS
130 READ DS:REM DOT SPACING
140 READ AS,AE:REM START AND END ANGLES FOR SEGMENT
150 IF DS=0 THEN 350
195 REM
200 REM DRAW SEGMENT
205 REM
210 D=DS*π/180
220 AS=AS*π/180
230 AE=AE*π/180
240 FOR R=DS TO RA STEP DS
250 FOR P=AS TO AE STEP D*(40/R)
260 X=R*COS(P)
270 Y=R*SIN(P)
280 X=.7*X+XC
290 Y=Y+YC
300 POINT 3,X,Y
310 NEXT P
320 NEXT R
330 GOTO 100
350 GETA$:IFA$=""THEN350
360 COLOR 1,3,6,0
370 GRAPHIC 0
380 END
395 REM
400 REM BORDER DRAWING ROUTINE
405 REM
410 POINT 3,0,0
420 DRAW 3 TO 1023,0
430 DRAW 3 TO 1023,1023
440 DRAW 3 TO 0,1023
450 DRAW 3 TO 0,0
460 RETURN
495 REM
```

```
500 DATA 50,1,70
510 DATA 20,71,200
520 DATA 30,201,300
530 DATA 10,301,360
540 DATA 0,0,0

READY.
```

GRAPH PLOTTING

GRAPH

DESCRIPTION

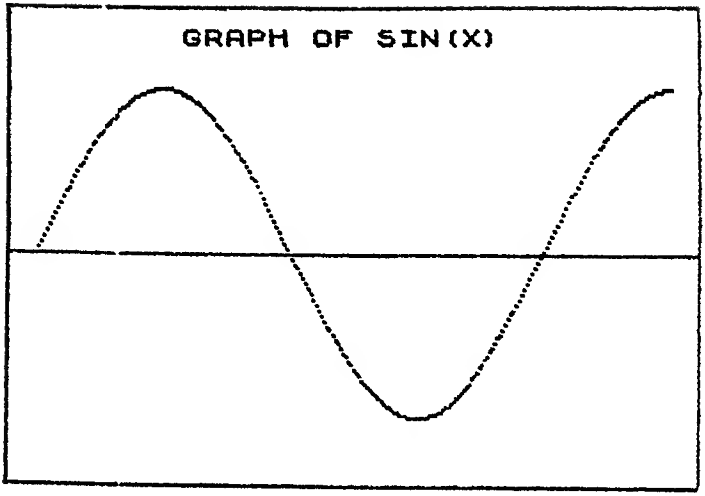
Each character position is made up of an 8 × 8 dot matrix, which means that we can plot points to a resolution of 176 in the X axis and 176 in the Y axis. The two programs GRAPH and GRAPH2 use the full resolution of the screen to plot respectively a graph of SIN (X) and SIN (X) with COS (X), using the VIC commands POINT, and DEF FN to define the function to be plotted. The programs are identical except for an additional routine in GRAPH2 to plot COS (X), and a couple of lines to identify the function and display a title.

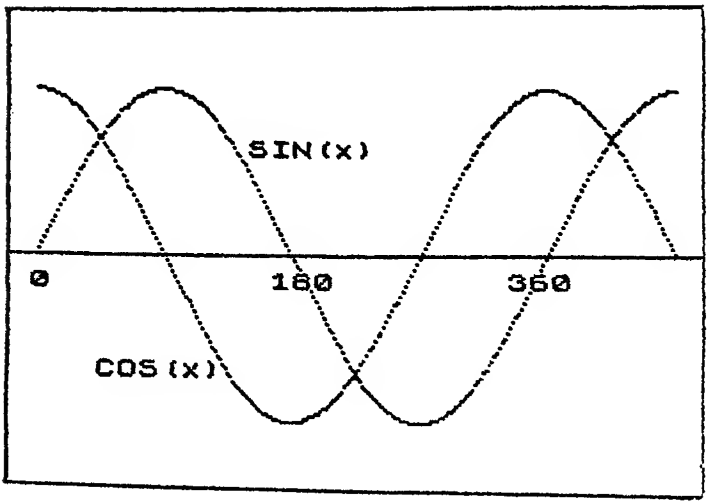
RUNNING THE PROGRAM

The INPUTting of variables is not required in either program, as we are simply taking the function a(x) to represent sin (x30)60 in the program GRAPH, and in addition b(x) to represent cos (x30)60 in the program GRAPH2. These are defined in line 130 in the former program, and lines 130-131 in the latter. It then runs through lines 200 to 330 to plot out the actual function. These routines could obviously be incorporated in further programs to plot different functions, just by altering the definitions in lines 130-131.

PROGRAM STRUCTURE

50-60	set colours
90	draw border around screen using subroutine at 400
130	define function(s) to be plotted
150-180	draw Y axis and label graph(s)
200-330	graph plotting routine
400-460	border drawing subroutine


GRAPH OF SIN(X)


SIN(x)
0
180
360
COS(x)

```
1 REM GRAPH
2 REM ***********************
3 REM
10 REM PROGRAM TO PLOT M THE GRAPH OF A FUNCTION
20 REM
30 REM SET COLOURS
40 REM
50 GRAPHIC 2
60 COLOR 3,3,0,10
70 REM
80 REM DRAW BORDER
85 REM
90 GOSUB 400
95 REM
100 REM DEFINE FUNCTION TO BE PLOTTED
105 REM
110 REM THE NUMERICAL VALUES IN THE EXAMPLE ARE
120 REM USED TO SCALE THE PLOT TO REASONABLE
125 REM DIMENSIONS
130 DEF FNA(X)=SIN(X/120)*400
135 REM
140 REM DRAW Y AXIS AT 0
145 REM
150 POINT 3,0,512
160 DRAW 3 TO 1023,512
180 CHAR 1,2,"GRAPH OF SIN(X)"
185 REM
190 REM PLOT GRAPH
195 REM
200 FOR X=0 TO 1023
210 Y=FNA(X)
220 POINT 3,X,512-Y
230 NEXT X
300 GETA$:IFA$=""THEN300
310 COLOR 1,3,6,0
320 GRAPHIC 0
330 END
395 REM
400 REM BORDER DRAWING ROUTINE
405 REM
410 POINT 3,0,0
420 DRAW 3 TO 0,1023
430 DRAW 3 TO 1023,1023
440 DRAW 3 TO 1023,0
450 DRAW 3 TO 0,0
460 RETURN
```

```
1 REM GRAPH 2
2 REM ***********************
3 REM
```

```
10 REM PROGRAM TO PLOT M THE GRAPH OF TWO FUNCTIONS
20 REM
30 REM SET COLOURS
40 REM
50 GRAPHIC 2
60 COLOR 3,3,0,10
70 REM
80 REM DRAW BORDER
85 REM
90 GOSUB 400
95 REM
100 REM DEFINE FUNCTION TO BE PLOTTED
105 REM
110 REM THE NUMERICAL VALUES IN THE EXAMPLE ARE
120 REM USED TO SCALE THE PLOT TO REASONABLE
125 REM DIMENSIONS
130 DEF FNA(X)=SIN(X/120)*400
131 DEF FNB(X)=COS(X/120)*400
135 REM
140 REM DRAW Y AXIS AT 0
145 REM
150 POINT 3,0,512
160 DRAW 3 TO 1023,512
170 CHAR 2,6,"SIN(X)"
180 CHAR 18,1,"COS(X)"
185 REM
190 REM PLOT GRAPH
195 REM
200 FOR X=0 TO 1023
210 Y=FNA(X)
220 POINT 3,X,512-Y
230 Y=FNB(X)
240 POINT 3,X,512-Y
250 NEXT X
260 CHAR 11,0,"0"
270 CHAR 11,6,"180"
280 CHAR 11,13,"360"
300 GETA$:IFA$=""THEN300
310 COLOR 1,3,6,0
320 GRAPHIC 0
330 END
395 REM
400 REM BORDER DRAWING ROUTINE
405 REM
410 POINT 3,0,0
420 DRAW 3 TO 0,1023
430 DRAW 3 TO 1023,1023
440 DRAW 3 TO 1023,0
450 DRAW 3 TO 0,0
460 RETURN
```

3D GRAPH

DESCRIPTION

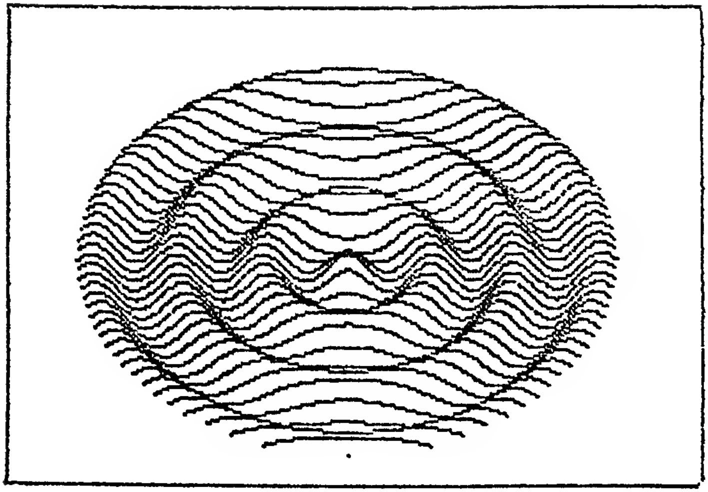
Building on from the routines for plotting two dimensional functions, we find that it is relatively easy to design a program for plotting in three dimensions. The program labelled 3D Graph does just that. Although we are relying on the same VIC Super Expander command POINT, our routine for plotting the function is, of necessity, rather more complicated this time, as we are trying to emulate a three dimensional image on what is, after all, a two dimensional screen. Of special interest in this routine is the double IF statement in line 310, which performs a straightforward RETURN depending on the values of the variables P and Z.

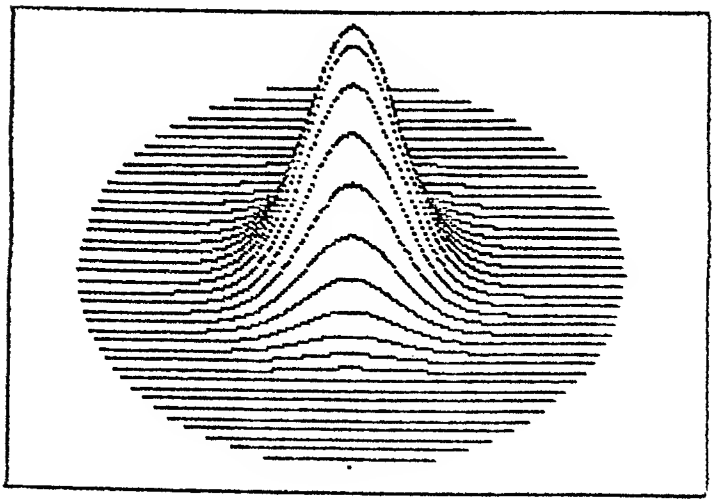
RUNNING THE PROGRAM

No variables are INPUT in this program, as our function is defined in line 150, and the area to be plotted in is determined by the scale given to X in line 220. This in turn determines the scale of Y to be plotted, by line 260. Line 270, the start of the inner of our two plotting loops, plots all the points on the Y axis for the value of X in the outer loop, which commences at line 220. We then move onto the next point on the X axis, and plot all the Y values there, and so on. By changing the definition in line 150 we can plot out a whole series of different functions.

PROGRAM STRUCTURE

60-70	set colours
90	draw border round screen using subroutine at 400
150	define function to be plotted
210-380	plotting routine
390-393	program termination
400-460	border drawing subroutine





```
1 REM 3D GRAPH
2 REM ************************
3 REM
10 REM THIS ROUTINE PLOTS THE GRAPH OF A
```

```
20 REM FUNCTION IN 3 DIMENSIONS
30 REM
40 REM
50 REM SET COLOURS
60 GRAPHIC 2
70 COLOR 3,3,0,10
80 REM DRAW BORDER AROUND SCREEN
90 GOSUB 400
95 REM
100 REM DEFINE FUNCTION TO BE PLOTTED
105 REM
110 REM THE FUNCTION IS CHANGED BY
115 REM ALTERING THE CONTENTS OF LINE 150
120 REM
150 DEF FNA(Z)=90*EXP(-Z*Z/600)
195 REM
200 REM PLOT GRAPH
205 REM
210 K=6
220 FOR X=-100 TO 0 STEP 1
230 L=0
240 P=1
250 Z1=0
260 Y1=K*INT(SQR(10000-X*X)/K)
270 FOR Y=Y1 TO -Y1 STEP -K
280 Z=INT(80+FNA(SQR(X*X+Y*Y))-.707106*Y)
290 IF Z<L THEN 350
295 GOSUB 380
300 L=Z
310 IF P=0 THEN GOSUB 380:IF Z=Z1 THEN GOSUB 380
320 POINT 3,.7*5*X+512,1023-5*Z
325 POINT 3,512-.7*5*X,1023-5*Z
330 IF P=0 THEN Z1=Z
340 P=0
350 NEXT Y
360 NEXT X
370 GOTO 390
380 RETURN
390 GETA$:IFA$=""THEN390
391 COLOR 1,3,6,0
392 GRAPHIC 0
393 END
395 REM
400 REM BORDER DRAWING ROUTINE
405 REM
410 POINT 3,0,0
420 DRAW 3 TO 1023,0
430 DRAW 3 TO 1023,1023
440 DRAW 3 TO 0,1023
450 DRAW 3 TO 0,0
460 RETURN
```

INTERPOLATE

DESCRIPTION

Determining a set of data is all very well, but it is the interpolation of that data that produces the all important results. One common method doing this is to take the data and turn it into points on a graph, and then perform the interpolation between those points. The program "Interpolate" does that, by assuming that you already have your data in the form of X, Y co-ordinates (here we store them as data statements in line 180), and plotting the appropriate point out within a defined area (lines 220 to 230 define the top and bottom of the Y axis and left and right of the X axis), before finally 'joining up' the points in whatever form you desire (see Running the Program). You could quite easily incorporate your own data into this program simply by changing the data statements in line 180.

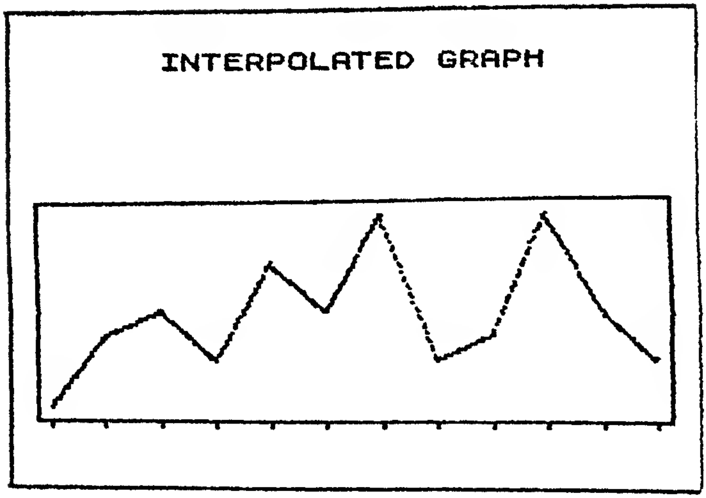
RUNNING THE PROGRAM

The main bulk of the work is done a) by the line 180, which stores the data as X, Y co-ordinates, and b) line 200, which determines which point we start at (here it is the first one), which one we finish at (here it is the twelfth), and which points we interpolate between (here it is every one, although by changing the variable SP in line 200 we could easily take every other point, for instance). Once we've calculated the scaling factors in lines 410 to 490, and turned these into point increments in lines 510 and 520, we plot the actual point in line 640, and the line between each point by the routine in lines 670-730.

PROGRAM STRUCTURE

60-70	set colours
90	draw border round screen using subroutine at 1000
110-160	read and store the data
180-181	data stored as X, Y co-ordinate
200	determine start and finish points, and separation
220-230	determine position and dimensions of graph on screen
310-380	draw border round graph, and label graph
410-490	determine scaling factors

510-520	convert scaling factors to point increments
610-720	point and line drawing routine
1000-1060	border drawing routine



```
1 REM INTERPOLATE
2 REM **********************
3 REM
10 REM PROGRAM TO DRAW A GRAPH BY INTERPOLATING
20 REM A SET OF POINTS STORED AS DATA STATEMENTS IN
30 REM LINE 180.
45 REM
50 REM SET COLOURS
55 REM
60 GRAPHIC 2
70 COLOR 3,3,0,10
75 REM
80 REM DRAW BORDER AROUND SCREEN
85 REM
90 GOSUB 1000
95 REM
100 REM INITIALISE DATA
105 REM
110 DIM X(12)
120 DIM Y(12)
130 FOR I=1 TO 12
140 READ X(I)
150 READ Y(I)
160 NEXT I
165 REM
170 REM DATA STORED AS X AND Y COORDINATE
175 REM
180 DATA1,10,2,25,3,30,4,20,5,40,6,30
181 DATA 7,50,8,20,9,25,10,50,11,30,12,20
185 REM
190 REM MIN DIMENSION =1, MAX =12, SEPERATION =1
195 REM
200 DN=1:DX=12:SP=1
205 REM
210 REM POSITION AND DIMENSIONS OF GRAPH ON SCREEN
215 REM
220 XL=75:XR=955
230 YB=900:YT=400
295 REM
300 REM DRAW BORDER AROUND GRAPH
305 REM
310 POINT 3,XR+10,YB+10
320 DRAW 3 TO XR+10,YT-10
330 DRAW 3 TO XL-10,YT-10
340 DRAW 3 TO XL-10,YB+10
350 DRAW 3 TO XR+10,YB+10
355 XI=(XR-XL)/(DX-DN)
360 FOR X=XL TO XR STEP XI
365 FOR A=10 TO 30
370 POINT 3,X,YB+A
375 NEXT A:NEXT X
```

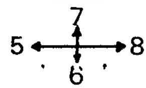
```
380 CHAR 1,1,"INTERPOLATED GRAPH"
395 REM
400 REM CALCULATE SCALING FACTORS
405 REM
410 Y1=-1000000
420 Y2=1000000
430 X1=Y1:X2=Y2
440 FOR I=DN TO DX STEP SP
450 IF Y1<Y(I) THEN Y1=Y(I)
460 IF Y2>Y(I) THEN Y2=Y(I)
470 IF X1<X(I) THEN X1=X(I)
480 IF X2>X(I) THEN X2=X(I)
490 NEXT I
495 REM
500 REM CONVERT SCALING FACTORS INTO POINT
505 REM INCREMENTS
510 A=(XR-XL)/(X1-X2)
520 B=(YB-YT)/(Y1-Y2)
595 REM
600 REM PLOT GRAPH
605 REM
610 FOR I=DN TO DX STEP SP
620 X=(XL+(X(I)-X2)*A)
630 Y=(YB-(Y(I)-Y2)*B)
660 POINT 3,X,Y
670 Q=I+SP
680 IF Q>DX THEN 900
690 X=(XL+(X(Q)-X2)*A)
700 Y=(YB-(Y(Q)-Y2)*B)
710 DRAW 3 TO X,Y
720 NEXT I
900 GETA$:IFA$=""THEN900
910 COLOR 1,3,6,0
920 GRAPHIC 0
930 END
1000 REM DRAW BORDER AROUND SCREEN
1005 REM
1010 POINT 3,0,0
1020 DRAW 3 TO 0,1023
1030 DRAW 3 TO 1023,1023
1040 DRAW 3 TO 1023,0
1050 DRAW 3 TO 0,0
1060 RETURN

READY.
```

USING THE VIDEO MEMORY

HI-RES CURSOR

DESCRIPTION

Many of the arcade games about at present require the movement of some kind of 'sight' around the screen, to get you to the right position before firing. Similarly, a routine to move a sight, or indeed a cursor over the screen, would have many uses in plotting, design, and graphic programs generally. The two programs here provide just such a routine, but achieved in slightly different manners. What they do have in common is the method of moving the cursor (here it is a cross) about, which uses the keys 5, 6, 7 and 8 in the following way:-



Thus, pressing the 8 key would move the cursor up, etc. This routine lies in lines 210 to 260. The two programs differ in that a) the cursor is designed differently in each one, and b) the first program erases whatever screen contents the cursor passes over: the second one doesn't.

RUNNING THE PROGRAM HI-RES CURSOR

Having drawn our border around the screen, the program positions the cursor at the X, Y coordinate of 24,24; and sets the increment between cursor movements (the variable S in line 120) to be 6. The program then simply waits until you press the appropriate key, increases or decreases X or Y accordingly, and then checks to see whether you are still within the screen boundary. If you are, it erases the previous cursor, draws a new one, and then awaits the pressing of another key.

PROGRAM STRUCTURE FOR HI-RES CURSOR

60-70	set colour
90	draw border round screen using subroutine at 600
110-120	set up parameters
210-260	check for key press
310-340	check if within boundary
410-440	erase previous cursor
460-490	draw new cursor
510	back to check for key press
600-660	border drawing subroutine

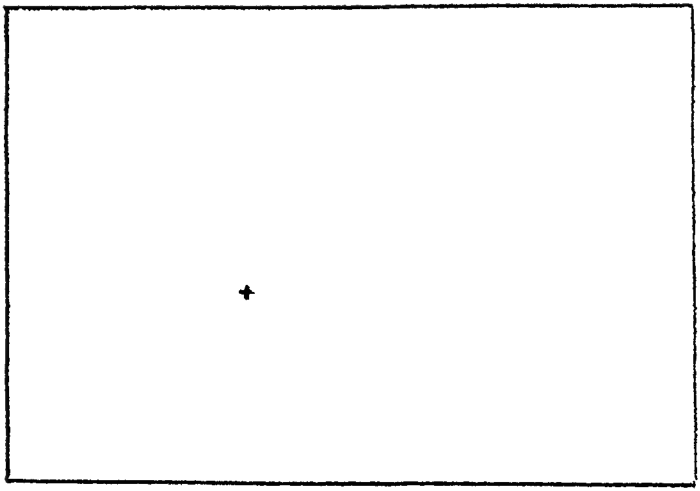
RUNNING THE PROGRAM HI-RES CURSOR 2

This follows roughly the same lines as Hi-Res Cursor, although our cursor is now defined in data statements contained in lines 150-170, and stored in the array C (5,5). This 'cursor' can now be anything you like, simply by changing the data statements. We plot the position of the cursor in lines 610-680. However, the point of this program is that we do NOT erase the screen contents, so the routine from lines 410 to 480 erases the cursor but then does an INVERSE on what has just gone, thus restoring the original screen display. Before plotting the cursor again we must save the screen contents into our array M(5,5), and this is performed by the function in lines 510 to 550, using the VIC command RDOT. This tells us what colour a certain position is, and we use this array again when going back to erase the cursor and re-trace the screen contents.

Different keys are used for cursor movement:
5: F3
6: F7
7: F1
8: F5

PROGRAM STRUCTURE FOR HI-RES CURSOR 2

60-70	set colours
90	fill the entire screen with characters
105-110	set up parameters
120-170	define 'cursor' and read data statements
210-260	wait for appropriate key press
310-345	check if within boundary
410-480	erase previous cursor and restore screen contents
510-550	save screen contents
610-680	plot new cursor
910	go back and wait for another key to be pressed



```
1 REM HI-RES CURSOR
2 REM ***********************
3 REM
10 REM PROGRAM TO MOVE A HIGH RESOLUTION CURSOR
20 REM ABOUT THE SCREEN UNDER CONTROL OF THE
30 REM KEYBOARD
40 REM
50 REM SET COLOURS
60 GRAPHIC 2
70 COLOR 3,3,0,3
75 REM
80 REM DRAW BORDER AROUND SCREEN
85 REM
90 GOSUB 600
95 REM
100 REM SET UP PARAMETERS
105 REM
110 X=24:Y=24:REM START POSITION
120 S=6:REM CURSOR MOVEMENT INCREMENTS
130 GOTO 450
195 REM
200 REM INPUT CURSOR MOVEMENT FROM KEYBOARD
205 REM
210 A=PEEK(197)
215 X0=X:Y0=Y
220 IF A=2 THEN X=X-S:GOTO 300
230 IF A=58 THEN Y=Y+S:GOTO 300
240 IF A=3 THEN Y=Y-S:GOTO 300
250 IF A=59 THEN X=X+S:GOTO300
```

```
260 GOTO 210
295 REM
300 REM CHECK CURSOR WITHIN BOUNDS
305 REM
310 IF X<24 THEN X=24
320 IF X>999 THEN X=999
330 IF Y<24 THEN Y=24
340 IF Y>999 THEN Y=999
395 REM
400 REM ERASE PREVIOUS CURSOR
405 REM
410 POINT 4,X0-12,Y0
420 DRAW 4 TO X0+12,Y0
430 POINT 4,X0,Y0-12
440 DRAW 4 TO X0,Y0+12
445 REM
450 REM PLOT NEW CURSOR
455 REM
460 POINT 3,X-12,Y
470 DRAW 3 TO X+12,Y
480 POINT 3,X,Y-12
490 DRAW 3 TO X,Y+12
495 REM
500 REM DO AGAIN
505 REM
510 GOTO 210
595 REM
600 REM DRAW BORDER AROUND SCREEN
605 REM
610 POINT 3,0,0
620 DRAW 3 TO 0,1023
630 DRAW 3 TO 1023,1023
640 DRAW 3 TO 1023,0
650 DRAW 3 TO 0,0
660 RETURN

READY.
```

```
1 REM HI-RES CURSOR 2
2 REM ***********************
3 REM
10 REM PROGRAM TO MOVE A HIGH RESOLUTION CURSOR ABOUT
20 REM THE SCREEN UNDER CONTROL OF THE KEYBOARD.
30 REM THE CURSOR DOES NOT ERASE EXISTING SCREEN
35 REM DISPLAYS.
40 REM
50 REM SET COLOURS
60 GRAPHIC 2
70 COLOR 3,3,0,3
```

```
75 REM
80 REM FILL SCREEN WITH CHARACTERS
85 REM
90 FOR I=0 TO 19
91 FOR J=0 TO 19
92 CHAR J,I,"■"
93 NEXT J
94 NEXT I
95 REM
100 REM SET UP PARAMETERS
105 X=5:Y=5:REM START POSITION
110 S=3:REM CURSOR MOVEMENT
120 DIM C(5,5),M(5,5)
125 FOR I=1 TO 5
130 FOR J=1 TO 5
135 READ C(J,I)
140 NEXT J
145 NEXT I
150 DATA 0,0,1,0,0
155 DATA 0,0,1,0,0
160 DATA 1,1,1,1,1
165 DATA 0,0,1,0,0
170 DATA 0,0,1,0,0
190 GOTO 500
195 REM
200 REM INPUT CURSOR MOVEMENT FROM KEYBOARD
205 REM
210 A=PEEK(197)
215 X0=X:Y0=Y
220 IF A=47 THEN X=X-S:GOTO 300
230 IF A=63 THEN Y=Y+S:GOTO 300
240 IF A=39 THEN Y=Y-S:GOTO 300
250 IF A=55 THEN X=X+S:GOTO300
260 GOTO 210
295 REM
300 REM CHECK CURSOR WITHIN BOUNDS
305 REM
310 IF X<5 THEN X=5
320 IF X>165 THEN X=165
330 IF Y<5 THEN Y=5
340 IF Y>165 THEN Y=165
395 REM
400 REM ERASE PREVIOUS CURSOR
405 REM
410 FOR I=-2 TO 2
420 FOR J=-2 TO 2
430 IF M(J+3,I+3)<>0 THEN 460
440 POINT 3,(J+X0)*6,(I+Y0)*6
445 REM
450 GOTO 470
455 REM
```

```
460 POINT 4,(J+X0)*6,(I+Y0)*6
470 NEXT J
480 NEXT I
495 REM
500 REM SAVE SCREEN CONTENTS
505 REM
510 FOR I=-2 TO 2
520 FOR J=-2 TO 2
530 M(J+3,I+3)=RDOT((J+X)*6,(I+Y)*6)
540 NEXT J
550 NEXT I
595 REM
600 REM PLOT NEW CURSOR
605 REM
610 FOR I=-2 TO 2
620 FOR J=-2 TO 2
630 IF C(J+3,I+3)=0 THEN 660
640 POINT 3,(X+J)*6,(Y+I)*6
650 GOTO 670
660 POINT 4,(X+J)*6,(I+Y)*6
670 NEXT J
680 NEXT I
895 REM
900 REM DO AGAIN
905 REM
910 GOTO 210

READY.
```

CHARACTER BUILDING

DESCRIPTION

Although there are numerous different graphics characters on the VIC, you may still want to define your own characters at times. This is easily done using the built-in character generator. The following program enables you to edit the existing characters using the cursor control keys. Note that this program will not run with the Super Expander cartridge in place.

The program is separated into two stages: Choosing the character you want to change, and editing that character.

RUNNING THE PROGRAM

To get a user defined character you must first move the character generator from the character generator ROM into the top end of RAM. Then the area of RAM being used must be protected by decreasing the end of memory pointer, as in lines 4-270. See the section on Hi-res for more information.

Having edited the character, you may return to see what it looks like, then go back and save it as a data statement. Or you may save it as a data statement as it is. Line 3000 is an example of how the data statement is formatted; the first value is the memory location and the next eight values are the values to go into memory from that location onwards. When you have finished editing, you can delete the rest of the program and have data that can be used in other programs.

PROGRAM STRUCTURE

4-270	initialise program by moving generator ROM into RAM
300	set line no. for data statements
330-340	define functions for screen poke location and value for character
360-390	print up grid for new char. option
400-480	wait for input from keyboard
510-610	cursor control options

710-770	define new character options
810-840	review character set options
1010-1170	display character set and options
1210-1300	display edit options
1510-1550	restore normal VIC operation mode and end
1610-1700	update edited character into character set
1810-1910	display character for editing in an 8x8 grid
2010-2160	add data statement on to end of program and re-run
3000	example data statement

```
1 REM CHARACTER BUILDING
2 REM ***********************
3 REM
4 POKE56,PEEK(56)-2
100 REM PET BENELUX
110 REM EXCHANGE
120 REM NETHERLANDS
125 REM
130 POKE 36879,42
140 PRINT"⊐▓* CHARACTER BUILDING *"
150 POKE 900,0
160 RUN 170
170 CS=256*PEEK(52)+PEEK(51)
180 FOR I=CS TO CS+511
190 POKE I,PEEK(32768+I-CS)
200 NEXT I
210 PRINT"⊐▚ RUN 280"
220 PRINT"RUN"
230 POKE 198,3
240 POKE 631,19
250 POKE 632,13
260 POKE 633,13
270 POKE 56,PEEK(56)+2:END
280 S=7680:CL=22
290 CS=256*PEEK(52)+PEEK(51)
300 CR=0:LN=3000+PEEK(900)
310 P=12:BO=1:BR=1
320 POKE 36879,42
330 DEFFNA(XX)=S+R*CL+C:REM SCREEN POKE LOCATION
340 DEFFNB(XX)=8*R+C:REM SCREEN POKE VALUE FOR CHAR
350 GOTO 1000
360 PRINT"⊐▚":GOSUB 1200
370 PRINT"▚";:FOR I=0 TO 7
380 PRINT"........"
390 NEXT I:F=0
400 PRINT"▚":R=0:C=0
410 Z=FNA(0)
420 IF F=0 THEN 460
430 IF Z=ZL THEN 450
440 POKE ZL,IL:ZL=Z:IL=PEEK(ZL)
450 POKE Z+30720,0
460 POKE Z+30720,0
470 GET A$:IF A$="" THEN 470
480 POKE Z+30720,1
490 REM
500 REM CURSOR CONTROL OPTIONS
505 REM
510 IF A$="Q" THEN 1500
520 IF A$="▐" AND C=7 THEN C=0:GOTO 410
530 IF A$="▐" THEN C=C+1:GOTO 410
540 IF A$="▌" AND C=0 THEN C=7:GOTO 410
```

```
550 IF A$="■" THEN C=C-1:GOTO 410
560 IF A$="■" AND R=7 THEN R=0:GOTO 410
570 IF A$="■" THEN R=R+1:GOTO 410
580 IF A$="□" AND R=0 THEN R=7:GOTO 410
590 IF A$="□" THEN R=R-1:GOTO 410
600 IF A$="■" THEN 400
610 IF F=1 THEN 800
695 REM
700 REM DEFINE NEW CHARACTER OPTIONS
705 REM
710 IF A$="+" THEN POKE Z,81:GOTO 410
720 IF A$="-" THEN POKE Z,46:GOTO 410
730 IF A$="=" THEN 1600
740 IF A$="□" THEN 370
750 IF A$="R" THEN 1000
760 IF A$="B" THEN 2000
770 GOTO 410
795 REM
800 REM REVIEW CHARACTER SET OPTIONS
805 REM
810 CR=FNB(0)
820 IF A$="N" THEN POKE 36869,240:GOTO 360
830 IF A$="E" THEN POKE 36869,240:F=0:GOTO 1800
840 GOTO 410
995 REM
1000 REM DISPLAY CHARACTER SET OPTIONS
1005 REM
1010 POKE 36869,255:R=4:C=0
1020 ZL=FNA(0):IL=32
1030 F=1:PRINT"□";
1040 PRINT"■@ABCDEFG"
1050 PRINT"HIJKLMNO"
1060 PRINT"PQRSTUVW"
1070 PRINT"XYZ[£]↑←"
1080 PRINT" !"CHR$(34)"#$%&'"
1090 PRINT"()*+,-./"
1100 PRINT"01234567"
1110 PRINT"89:;<=>?"
1120 PRINT"■"SPC(13)"■OPTIONS■"
1130 PRINTSPC(10)"■N NEW CHAR■"
1140 PRINTSPC(10)"■E EDIT CHAR■"
1150 PRINTSPC(10)"■Q QUIT■"
1160 BC=PEEK(38400)
1170 GOTO 410
1195 REM
1200 REM EDIT OPTIONS
1205 REM
1210 PRINT"■■"SPC(13)"■OPTIONS■"
1220 PRINT
1230 PRINTSPC(P)"■+■ ADD DOT"
1240 PRINTSPC(P)"■-■ ERASE"
```

```
1250 PRINTSPC(P)"▓=■ UPDATE"
1260 PRINTSPC(P)"▓R■ REVIEW"
1270 PRINTSPC(P)"▓Q■ QUIT"
1280 PRINTSPC(P)"▓B■ ADD DATA"
1290 PRINTSPC(P+1)"▖STATEMENT"
1300 RETURN
1495 REM
1500 REM QUIT
1505 REM
1510 POKE 56,PEEK(56)+2
1520 POKE 36869,240
1530 POKE 36879,27
1540 PRINT"▖▓     BYE!"
1550 END
1595 REM
1600 REM UPDATE
1605 REM
1610 PRINT"▓";
1620 X=CS+8*CR
1630 FOR R=0 TO 7:SM=0
1640 FOR C=0 TO 7:D=7-C
1650 SM=SM-2↑D*(PEEK(FNA(0))=81)
1660 NEXT C
1670 POKE X+R,SM
1680 PRINTSPC(8);SM
1690 NEXT R:R=0:C=0
1700 GOTO 410
1795 REM
1800 REM EDIT CHAR
1805 REM
1810 PRINT"▖"
1820 X=CS+8*CR
1830 FOR R=0 TO 7:Y=PEEK(X+R)
1840 FOR C=0 TO 7:Z=FNA(0)
1850 Q=46:Y=Y*2
1860 IF Y>255 THEN Q=81:Y=Y-256
1870 POKE Z,Q
1880 NEXT C,R
1890 R=0:C=0
1900 GOSUB 1200
1910 GOTO 410
1995 REM
2000 REM ADD DATA STATEMENTS
2005 REM
2010 X=CS+8*CR
2020 PRINT"▖▓▓▓▓▓▓▓▓▓▓"
2030 PRINTLN;"DATA";
2040 PRINTRIGHT$(STR$(X),LEN(STR$(X))-1);
2050 FOR I=X TO X+7
2060 PRINT",";
2070 PRINTRIGHT$(STR$(PEEK(I)),LEN(STR$(PEEK(I)))-1);
```

```
2080 NEXT I
2090 PRINT:PRINT"RUN S"
2100 POKE 900,PEEK(900)+1
2110 POKE 198,9
2120 FOR I=0 TO 8
2130 POKE I+631,13
2140 NEXT I
2150 POKE 56,PEEK(56)+2
2160 END
3000 DATA7472,48,72,72,48,74,68,58,0

READY.
```

BIG CHARACTER

DESCRIPTION

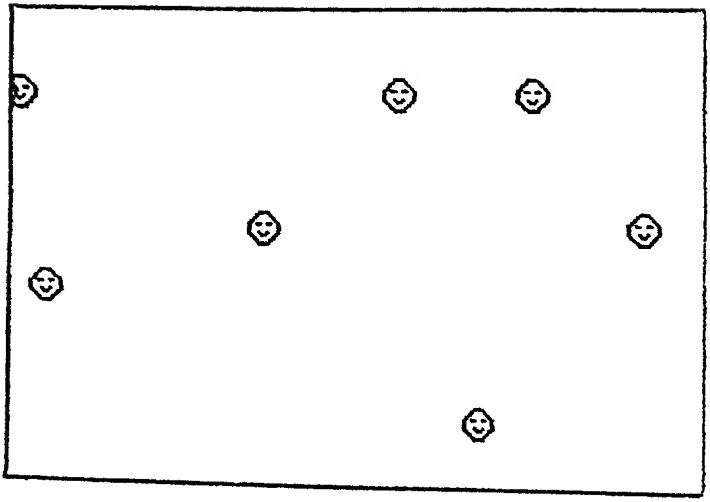
The program Big Character displays the use of the POINT command. This enables us to plot points to the full resolution of the VIC, that is 160 pixels by 160 pixels. The routine shown here, in lines 220 to 270, could be used in any program where we require a character that has previously been defined with the use of data statements, to be displayed on the screen at a specified central X,Y coordinate.

RUNNING THE PROGRAM

The data for our large character is stored in the data statements in lines 1100 to 1180. The first two numbers define the size of the character array which we will use to store the data; note that this is dynamically dimensionted on reading that data. Here we are storing the information in binary form: that is, using the digits 0 and 1 to define whether a particular pixel is to be 'lit' or 'unlit'. If you hold the book far enough away from you, you can probably see the character actually drawn out by those data statements. Having stored all the information in the array C(X,Y), we input the variables XC and YC to define the central coordinate for displaying the character, and finally the routine in lines 220 to 270 plots out the character on the screen. Line 300 then sends us back to plot out another one, and so on.

PROGRAM STRUCTURE

60-70	set colours
90	draw border round screen using subroutine at 500
120-180	set up character array from data statements
210-219	input character centre coordinates
220-270	plot character on screen
300	back again for another go
500-560	border drawing subroutine
1000-1060	input routine
1100-1180	data statements for character



```
1 REM BIG CHARACTER
2 REM ***********************
3 REM
10 REM THIS PROGRAM GENERATES LARGE CHARACTERS USING
20 REM THE POINT COMMAND, WITH CHARACTER DATA
30 REM STORED IN AN ARRAY
40 REM
50 REM SET COLOURS
60 GRAPHIC 2
70 COLOR 3,3,0,10
75 REM
80 REM DRAW BORDER
85 REM
90 GOSUB 500
95 REM
100 REM SET UP CHARACTER ARRAY FROM DATA STATEMENTS
110 REM
120 READ X,Y
130 DIM C(X,Y)
140 FOR I=1 TO Y
150 FOR J=1 TO X
160 READ C(J,I)
170 NEXT J
180 NEXT I
190 REM
200 REM INPUT CHARACTER COORDINATES AND DRAW
205 REM CHARACTER
210 REM INPUT CHARACTER CENTRE COORDINATES
211 Z$="":J=5
212 CHAR 19,2,"?"
213 GOSUB 1000
214 XC=A:Z$=""
215 CHAR 19,J,",":J=J+1
216 GOSUB 1000
217 YC=A:FOR I=1 TO 500:NEXT I
218 IF XC<0 OR YC<0 THEN 350
219 CHAR 19,2,"                 "
220 FOR I=1 TO Y
230 FOR J=1 TO X
240 IF C(J,I)=0 THEN 260
241 IF XC+6*(J-X/2)<0 THEN 260
242 IF YC+6*(I-Y/2)<0 THEN 260
243 IF YC+6*(I-Y/2)>950 THEN 260
250 POINT 3,XC+6*(J-Y/2),YC+6*(I-X/2)
260 NEXT J
270 NEXT I
280 RESTORE
300 GOTO 200
350 REM END
360 COLOR 1,3,6,0
370 GRAPHIC 0
```

```
380 END
495 REM
500 REM DRAW BORDER AROUND SCREEN
505 REM
510 POINT 3,0,0
520 DRAW 3 TO 1023,0
530 DRAW 3 TO 1023,950
540 DRAW 3 TO 0,950
550 DRAW 3 TO 0,0
560 RETURN
995 REM
1000 REM ROUTINE TO INPUT DATA
1005 REM
1010 GET A$:IF A$="" THEN 1010
1020 IF (ASC(A$)<48 OR ASC(A$)>57) AND A$<>"-" THEN 1050
1030 CHAR 19,J,A$:J=J+1
1040 Z$=Z$+A$:GOTO 1010
1050 A=VAL(Z$)
1060 RETURN
1095 REM
1100 DATA 12,12
1102 DATA 0,0,0,0,1,1,1,1,0,0,0,0
1105 DATA 0,0,0,1,0,0,0,0,1,0,0,0
1107 DATA 0,0,1,0,0,0,0,0,0,1,0,0
1108 DATA 0,1,0,0,0,0,0,0,0,0,1,0
1110 DATA 1,0,0,1,1,0,0,1,1,0,0,1
1120 DATA 1,0,0,0,0,0,0,0,0,0,0,1
1130 DATA 1,0,0,0,0,0,0,0,0,0,0,1
1140 DATA 1,0,0,0,1,0,0,1,0,0,0,1
1150 DATA 0,1,0,0,0,1,1,0,0,0,1,0
1160 DATA 0,0,1,0,0,0,0,0,0,1,0,0
1170 DATA 0,0,0,1,0,0,0,0,1,0,0,0
1180 DATA 0,0,0,0,1,1,1,1,0,0,0,0

READY.
```

SCALING and STRETCHING

SCALE 1

DESCRIPTION

The ability to scale a shape is one of the most useful in the computer's repertoire, and finds a home in many a program. For instance, computer aided design would not be where it is today without this function. Unfortunately, the VIC does not have a scaling command, and hence this routine. In its most simple form as we present it here, scaling just involves taking an object (here we have a rather simplistic view of a tree!), increasing the size of each line that makes up the object, and plotting out our new drawing. What this particular program suffers from is movement of the object as new ones are plotted: in other words, our original design does not get surrounded by larger ones, or itself surrounds smaller ones, but just becomes part of a grand row of small, medium and large trees.

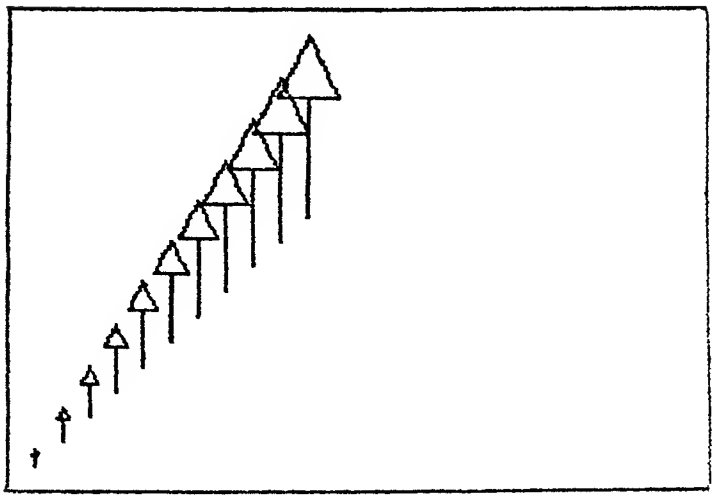
RUNNING THE PROGRAM

In line 110 we dimension our shape data arrays to contain 20 variables each. The data comes from the statements in lines 210 to 250, and as you can see the first number read is the number of sets of data statements to come: in our case 4. Dimensioning to 20 is just a precaution! In order, the data statements present the coordinates X, Y of the start of one of the lines that make up the tree, and the coordinates of the end of that line. Hence, four statements for our four line drawing. The scaling factor S is then input in line 280: when S ⅜ 1 we have the original size, a number less than 1 is smaller, and a number greater than 1 gives us a larger image. Scaling factors are then calculated in lines 310 to 360, and our new image plotted out in lines 410 to 530, by drawing out each line in turn. Our usual variable DS is used for dot spacing, and you can specify this to be whatever you like. As pointed our earlier, this program suffers from not having a constant central coordinate.

PROGRAM STRUCTURE

60-70	set colours
90	draw border using subroutine at 800
110-120	set up shape and scaled shape data arrays
140-170	read data for shape
210-250	data for shape

280-285	input scaling factor
310-360	calculate scaling
410-530	plot each line in turn to specified size
600	go back for another go with a new scaling factor
800-860	border drawing subroutine
900-930	end
1000-1110	input routine



```
1 REM SCALE 1
2 REM **********************
3 REM
10 REM ROUTINE TO CHANGE THE SCALE OF A SHAPE IN
20 REM THE SHAPE DATA TABLE
30 REM
40 REM
50 REM SET COLOURS
60 GRAPHIC 2
70 COLOR 3,3,0,10
75 REM
80 REM DRAW BORDER AROUND SCREEN
85 REM
90 GOSUB 800
95 REM
100 REM SET UP AND INPUT DATA FOR SCALING
105 REM
106 REM SHAPE DATA ARRAYS
107 REM
110 DIM X(20),Y(20),U(20),V(20)
115 REM
116 REM SCALED SHAPE DATA ARRAY
117 REM
120 DIM A(20),B(20),C(20),D(20)
125 REM
130 REM SET UP SHAPE DATA ARRAY
135 REM
140 READ NL:REM NUMBER OF LINES IN SHAPE
150 FOR I=1 TO NL
160 READ X(I),Y(I),U(I),V(I)
170 NEXT I
200 REM SHAPE DATA
205 REM
210 DATA 4
220 DATA 100,90,100,130
230 DATA 100,150,90,130
240 DATA 90,130,110,130
250 DATA 110,130,100,150
260 REM
280 REM INPUT SCALING FACTOR
281 Z$="":T=5
282 CHAR 19,2,"?                 "
283 GOSUB 1000
284 S=Z
285 IF S=0 THEN 900
290 REM
300 REM DO SCALING
305 REM
310 FOR I=1 TO NL
320 A(I)=X(I)*S
330 B(I)=Y(I)*S
```

```
340 C(I)=U(I)*S
350 D(I)=V(I)*S
360 NEXT I
395 REM
400 REM DRAW SHAPE
405 REM
410 FOR J=1 TO NL
420 DS=1
430 P=C(J)-A(J)
440 Q=D(J)-B(J)
450 R=SQR(P*P+Q*Q)
460 LX=P/R
470 LY=Q/R
480 FOR I=0 TO R STEP DS
490 X=6*(A(J)+I*LX)
500 Y=950-6*(B(J)+I*LY)
502 IF X<0 OR Y<0 THEN 520
504 IF X>1023 OR Y>950 THEN 520
510 POINT 3,X,Y
520 NEXT I
530 NEXT J
600 GOTO 280:REM DO AGAIN
795 REM
800 REM DRAW BORDER AROUND SCREEN
805 REM
810 POINT 3,0,0
820 DRAW 3 TO 1023,0
830 DRAW 3 TO 1023,950
840 DRAW 3 TO 0,950
850 DRAW 3 TO 0,0
860 RETURN
900 REM END PROGRAM
910 COLOR 1,3,6,0
920 GRAPHIC 0
930 END
995 REM
1000 REM INPUT DATA
1005 REM
1010 GET A$:IF A$="" THEN 1010
1020 IF (ASC(A$)<48 OR ASC(A$)>57) AND A$<>"." THEN 1100
1030 Z$=Z$+A$
1035 CHAR 19,T,A$
1040 T=T+1
1050 GOTO 1010
1100 Z=VAL(Z$)
1110 RETURN

READY.
```

SCALE 2

DESCRIPTION

Again here we are taking a shape, and scaling it in both X and Y directions, but with the major fault of the previous program rectified. This time we have a routine to correct the movement of the object as it is scaled, and plot everything out from a common, constant X,Y coordinate. Thus we have the same shape, expanded in both X and Y, or indeed contracted in X and Y, all centred on the same coordinates. This new routine is quite a straightforward 7 line one (lines 310 to 350). One other difference is that our object is rather more exotic this time, being made up of six lines rather than just 4. You can of course experiment with objects that are far more complicated than this: just be careful about the data statements in lines 210 to 255, and make sure you have all the X,Y coordinates right, and more importantly in the right order.

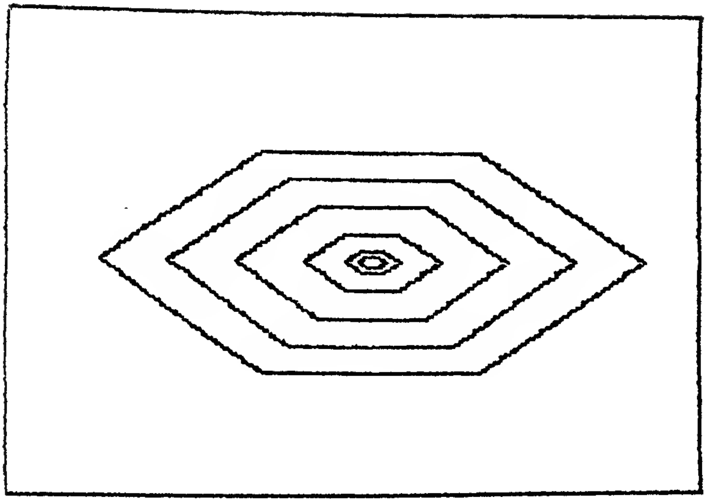
RUNNING THE PROGRAM

As with Scale 1, we dimension our shape and scaled shape data arrays (lines 110 to 120), read in the shape data (lines 140 to 170), and give the data statements (lines 210 to 255). The scaling factor S is input in line 280: as before a number greater than 1 means a larger shape, and less than 1 means a smaller one. The illustration shown ranges from S = 1.5 down to S = 0.1. The same routines as previously used are here to perform the scaling and draw the shape. The only new one is contained in lines 310 to 350, which calculates the central coordinates for our larger (or smaller) object: these are the variables CX and CY.

PROGRAM STRUCTURE

60-70	set colours
90	draw border using subroutine at 800
110-120	set up shape and scaled shape data arrays
140-170	read data for shape
210-255	data for shape
280-285	input scaling factor
310-350	calculate new central coordinates
410-460	calculate scaling
510-630	plot each line in turn to specified size
700	go back for another go with a new scaling factor
800-860	border drawing subroutine

900-930 end
1000-1110 input routine



```
1 REM SCALE 2
2 REM ***********************
3 REM
10 REM ROUTINE TO CHANGE THE SCALE OF A SHAPE IN
20 REM THE SHAPE DATA TABLE
30 REM USING THE SHAPES CENTRE
40 REM
50 REM SET COLOURS
60 GRAPHIC 2
70 COLOR 3,3,0,10
```

```
75 REM
80 REM DRAW BORDER AROUND SCREEN
85 REM
90 GOSUB 800
95 REM
100 REM SET UP AND INPUT DATA FOR SCALING
105 REM
106 REM SHAPE DATA ARRAYS
107 REM
110 DIM X(20),Y(20),U(20),V(20)
115 REM
116 REM SCALED SHAPE DATA ARRAY
117 REM
120 DIM A(20),B(20),C(20),D(20)
125 REM
130 REM SET UP SHAPE DATA ARRAY
135 REM
140 READ NL:REM NUMBER OF LINES IN SHAPE
150 FOR I=1 TO NL
160 READ X(I),Y(I),U(I),V(I)
170 NEXT I
200 REM SHAPE DATA
205 REM
210 DATA 6
220 DATA 100,110,140,110
230 DATA 140,110,170,90
235 DATA 170,90,140,70
240 DATA 140,70,100,70
250 DATA 100,70,70,90
255 DATA 70,90,100,110
260 REM
280 REM INPUT SCALING FACTOR
281 Z$="":T=5
282 CHAR 19,2,"?                    "
283 GOSUB 1000
284 S=Z
285 IF S=0 THEN 900
290 REM
300 REM FIND CENTRE
305 REM
310 CX=0:CY=0
320 FOR J=1 TO NL
330 CX=CX+X(J)+U(J)
335 CY=CY+Y(J)+V(J)
340 NEXT J
345 CX=CX/(2*NL)
350 CY=CY/(2*NL)
395 REM
400 REM DO SCALING
405 REM
410 FOR J=1 TO NL
```

```
420 A(J)=CX+(CX-X(J))*S
430 B(J)=CY+(CY-Y(J))*S
440 C(J)=CX+(CX-U(J))*S
450 D(J)=CY+(CY-V(J))*S
460 NEXT J
495 REM
500 REM DRAW SHAPE
505 REM
510 FOR J=1 TO NL
520 DS=1
530 P=C(J)-A(J)
540 Q=D(J)-B(J)
550 R=SQR(P*P+Q*Q)
560 LX=P/R
570 LY=Q/R
580 FOR I=0 TO R STEP DS
590 X=6*(A(J)+I*LX-40)
600 Y=950-6*(B(J)+I*LY)
602 IF X<0 OR Y<0 THEN 620
604 IF X>1023 OR Y>950 THEN 620
610 POINT 3,X,Y
620 NEXT I
630 NEXT J
700 GOTO 280:REM DO AGAIN
795 REM
800 REM DRAW BORDER AROUND SCREEN
805 REM
810 POINT 3,0,0
820 DRAW 3 TO 1023,0
830 DRAW 3 TO 1023,950
840 DRAW 3 TO 0,950
850 DRAW 3 TO 0,0
860 RETURN
900 REM END PROGRAM
910 COLOR 1,3,6,0
920 GRAPHIC 0
930 END
995 REM
1000 REM INPUT DATA
1005 REM
1010 GET A$:IF A$="" THEN 1010
1020 IF (ASC(A$)<48 OR ASC(A$)>57) AND A$<>"." THEN 1100
1030 Z$=Z$+A$
1035 CHAR 19,T,A$
1040 T=T+1
1050 GOTO 1010
1100 Z=VAL(Z$)
1110 RETURN

READY.
```

STRETCH 1

DESCRIPTION

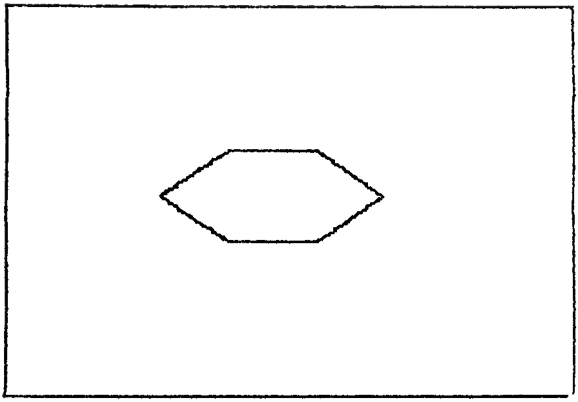
Stretching, although on the surface the same thing as scaling, is in fact a very different animal. Scaling merely produces a larger or smaller image of our original object, based either around the same or a different central coordinate. Stretching, on the other hand, does not necessarily change every line of our object to the same extent, but ideally we do want to stick to the same central coordinates. You can see in the illustrations here that we have a normal image, one stretched in the X axis, and one stretched in the Y axis. With the program being written the way that it has, you can combine stretching in both X and Y axes, without having to use the same stretching factor for each one.

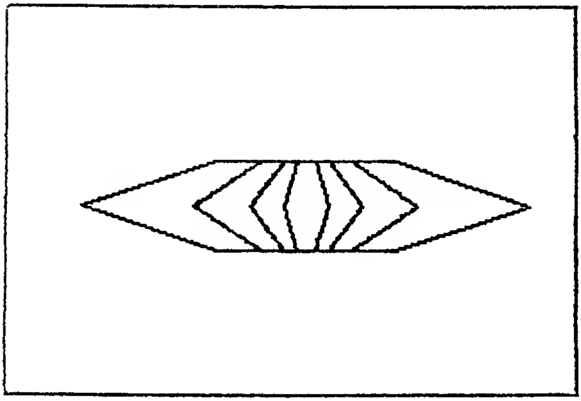
RUNNING THE PROGRAM

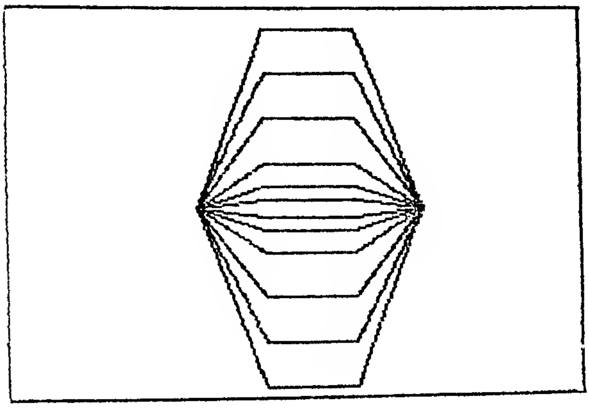
Until we reach line 280 the program follows the same lines as our earlier Scale 2 program. That is, we set up our shape and scaled shape data arrays (lines 110 to 120), and read in the data in lines 210 to 255 by the routine in lines 140 and 170. You will note that we are using the same object as last time, that is, a six sided figure. Lines 280-288 let us input the scaling factors SX and SY in the X and Y axes, and these are later used in lines 410 and 460 to calculate the scaling and stretching figures. Before and after that we find the central coordinates of our object (lines 310 to 350), and actually plot the figure out (lines 510 to 630) one line at a time.

PROGRAM STRUCTURE

60-70	set colours
90	draw border using subroutine at 800
110-120	set up shape and scaled shape data arrays
140-170	read data for shape
210-255	data for shape
280-288	input scaling factors
310-350	calculate new central coordinates
410-460	calculate scaling
510-630	plot each line in turn to specified size
700	go back for another go with a new scaling factor
800-860	border drawing subroutine
900-930	end
1000-1110	input routine







```
1 REM STRETCH 1
2 REM ***********************
3 REM
10 REM ROUTINE TO STRETCH OR CHANGE THE SCALE OF A SHAPE
20 REM IN THE SHAPE DATA TABLE. IT USES THE SHAPES
30 REM CENTRE AND DIFFERENTIAL X,Y,SCALING FACTORS.
40 REM
50 REM SET COLOURS
60 GRAPHIC 2
70 COLOR 3,3,0,10
75 REM
80 REM DRAW BORDER AROUND SCREEN
85 REM
90 GOSUB 800
95 REM
100 REM SET UP AND INPUT DATA FOR SCALING
105 REM
106 REM SHAPE DATA ARRAYS
107 REM
110 DIM X(20),Y(20),U(20),V(20)
115 REM
116 REM SCALED SHAPE DATA ARRAY
117 REM
120 DIM A(20),B(20),C(20),D(20)
125 REM
130 REM SET UP SHAPE DATA ARRAY
135 REM
140 READ NL:REM NUMBER OF LINES IN SHAPE
150 FOR I=1 TO NL
160 READ X(I),Y(I),U(I),V(I)
170 NEXT I
200 REM SHAPE DATA
205 REM
210 DATA 6
220 DATA 100,110,140,110
230 DATA 140,110,170,90
235 DATA 170,90,140,70
240 DATA 140,70,100,70
250 DATA 100,70,70,90
255 DATA 70,90,100,110
260 REM
280 REM INPUT SCALING FACTORS IN X AND Y AXIS
281 Z$="":T=5
282 CHAR 19,2,"?                    "
283 GOSUB 1000
284 SX=Z
285 CHAR 19,T,",":T=T+1
286 Z$="":Z=0:GOSUB 1000
287 SY=Z
288 IF SX=0 OR SY=0 THEN 900
290 REM
```

```
300 REM FIND CENTRE
305 REM
310 CX=0:CY=0
320 FOR J=1 TO NL
330 CX=CX+X(J)+U(J)
335 CY=CY+Y(J)+V(J)
340 NEXT J
345 CX=CX/(2*NL)
350 CY=CY/(2*NL)
395 REM
400 REM DO SCALING AND STRETCHING
405 REM
410 FOR J=1 TO NL
420 A(J)=CX+(CX-X(J))*SX
430 B(J)=CY+(CY-Y(J))*SY
440 C(J)=CX+(CX-U(J))*SX
450 D(J)=CY+(CY-V(J))*SY
460 NEXT J
495 REM
500 REM DRAW SHAPE
505 REM
510 FOR J=1 TO NL
520 DS=1
530 P=C(J)-A(J)
540 Q=D(J)-B(J)
550 R=SQR(P*P+Q*Q)
560 LX=P/R
570 LY=Q/R
580 FOR I=0 TO R STEP DS
590 X=6*(A(J)+I*LX-40)
600 Y=950-6*(B(J)+I*LY)
602 IF X<0 OR Y<0 THEN 620
604 IF X>1023 OR Y>950 THEN 620
610 POINT 3,X,Y
620 NEXT I
630 NEXT J
700 GOTO 280:REM DO AGAIN
795 REM
800 REM DRAW BORDER AROUND SCREEN
805 REM
810 POINT 3,0,0
820 DRAW 3 TO 1023,0
830 DRAW 3 TO 1023,950
840 DRAW 3 TO 0,950
850 DRAW 3 TO 0,0
860 RETURN
900 REM END PROGRAM
910 COLOR 1,3,6,0
920 GRAPHIC 0
930 END
995 REM
```

```
1000 REM INPUT DATA
1005 REM
1010 GET A$:IF A$="" THEN 1010
1020 IF (ASC(A$)<48 OR ASC(A$)>57) AND A$<>"." THEN 1100
1030 Z$=Z$+A$
1035 CHAR 19,T,A$
1040 T=T+1
1050 GOTO 1010
1100 Z=VAL(Z$)
1110 RETURN

READY.
```

STRETCH 2

DESCRIPTION

The Stretch 1 program as described is an extremely useful one, but alas it is not without its limitations. Although we can stretch images in both X and Y directions, one thing which we do not have control over is the angle of stretching. At present, everything is going at ninety degree angles. What if, as is very common in computer aided design, and indeed other fields, we want to stretch something at, say, 37 degrees to the X axis? The routine in lines 410 to 650 in this program performs just that function. I will not go into the mathematical detail here, many excellent books have been written on the subject, but will simply say that it works!

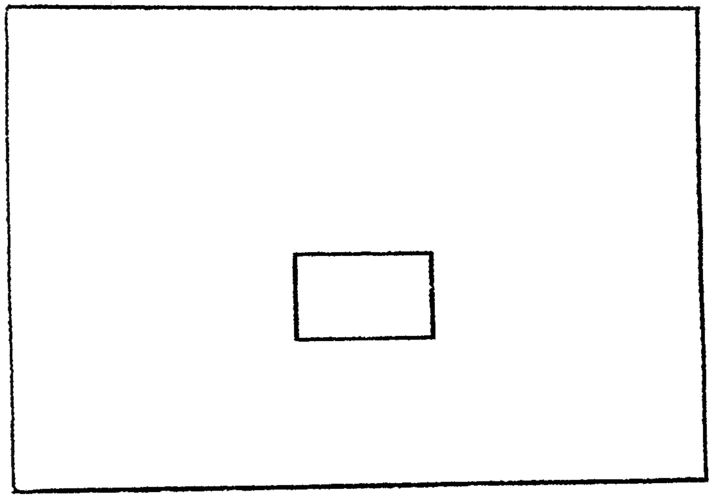
RUNNING THE PROGRAM

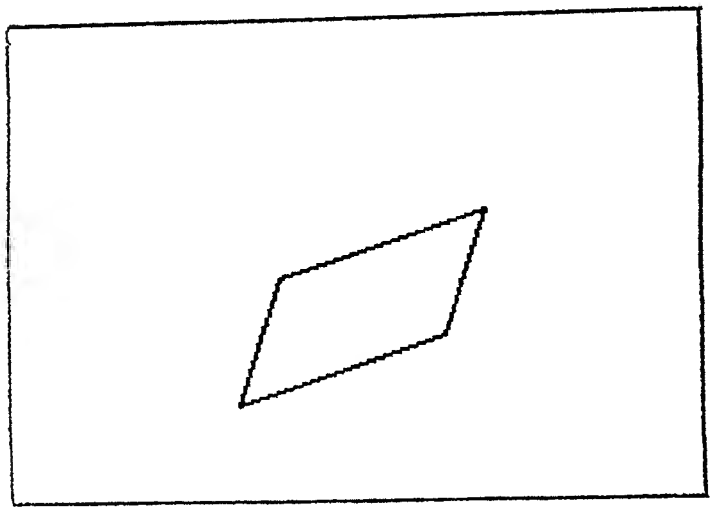
As in previous programs, we first of all set up the shape and scaled shape data arrays before reading in the actual data itself from lines 210 to 240. This time we revert to a much simpler shape, that of a rectangle. In line 280 we again input the scaling factors in the X and Y axes, and in line 290 we input AS, the angle of stretching. This is the angle by which we will evaluate our shape above the X axis. In other words, if AS is equal to 45 degrees, as it is in the illustration, the line joining the two corners of the rectangle will be at 45 degrees to the X axis. After calculating the centre of the newly formed shape, the scaling, stretching and rotating routine in lines 410 to 650 comes into effect. As you can see this is quite complicated, and I do not intend to go into any detail. This book is designed to help you with graphics on the VIC, not give you a thesis on mathematical theory! The program needs a 16K expansion RAM.

PROGRAM STRUCTURE

60-70	set colours
90	draw border using routine at 1000
110-120	dimension shape and scaled shape data arrays
140-170	read shape data
210-240	shape data statements
280-288	input scaling factors in X and Y axis
289-293	input angle of rotation; convert degrees to radians

310-350	calculate centre coordinates
410-650	perform scaling, stretching and rotation calculations
710-830	draw new shape line by line
840	go back for another go
1000-1060	border drawing subroutine
1100-1160	input routine





```
1 REM STRETCH 2
2 REM **********************
3 REM
10 REM ROUTINE TO STRETCH OR CHANGE THE SCALE OF A SHAPE
20 REM IN THE SHAPE DATA TABLE. IT USES THE SHAPES
30 REM CENTRE AND DIFFERENTIAL X,Y,SCALING FACTORS.
40 REM PLUS AN ANGLE OF ROTATION ALONG WHICH STRETCHING
45 REM TAKES PLACE.
46 REM
50 REM SET COLOURS
60 GRAPHIC 2
70 COLOR 3,3,0,10
75 REM
80 REM DRAW BORDER AROUND SCREEN
85 REM
90 GOSUB 1100
95 REM
100 REM SET UP AND INPUT DATA FOR SCALING
105 REM
106 REM SHAPE DATA ARRAYS
107 REM
110 DIM X(20),Y(20),U(20),V(20)
115 REM
116 REM SCALED SHAPE DATA ARRAY
117 REM
120 DIM A(20),B(20),C(20),D(20)
125 REM
130 REM SET UP SHAPE DATA ARRAY
135 REM
140 READ NL:REM NUMBER OF LINES IN SHAPE
150 FOR I=1 TO NL
160 READ X(I),Y(I),U(I),V(I)
170 NEXT I
200 REM SHAPE DATA
205 REM
210 DATA 4
220 DATA 100,120,150,120
230 DATA 150,120,150,90
235 DATA 150,90,100,90
240 DATA 100,90,100,120
260 REM
280 REM INPUT SCALING FACTORS IN X AND Y AXIS
281 Z$="":T=5
282 CHAR 19,2,"?                    "
283 GOSUB 1000
284 SX=Z
285 CHAR 19,T,",":T=T+1
286 Z$="":Z=0:GOSUB 1000
287 SY=Z
288 IF SX=0 OR SY=0 THEN 900
289 REM INPUT ANGLE OF STRETHCING
```

```
290 Z$="":T=T+5
291 CHAR 19,T-3,"?"
292 GOSUB 1000
293 AS=Z*π/180
295 REM
300 REM FIND CENTRE
305 REM
310 CX=0:CY=0
320 FOR J=1 TO NL
330 CX=CX+X(J)+U(J)
335 CY=CY+Y(J)+V(J)
340 NEXT J
345 CX=CX/(2*NL)
350 CY=CY/(2*NL)
395 REM
400 REM DO SCALING AND STRETCHING
405 REM
410 FOR I=1 TO NL
420 X1=X(I)-CX
430 Y1=Y(I)-CY
440 F=(X1*COS(AS)+Y1*SIN(AS))*SY
450 G=(-X1*SIN(AS)+Y1*COS(AS))*SX
460 X2=F*COS(AS)-G*SIN(AS)
470 A(I)=X2+CX
480 Y2=F*SIN(AS)+G*COS(AS)
490 B(I)=Y2+CY
500 X1=U(I)-CX
510 Y1=V(I)-CY
520 F=(X1*COS(AS)+Y1*SIN(AS))*SY
530 G=(-X1*SIN(AS)+Y1*COS(AS))*SX
540 X2=F*COS(AS)-G*SIN(AS)
550 C(I)=X2+CX
560 Y2=F*SIN(AS)+G*COS(AS)
570 D(I)=Y2+CY
650 NEXT I
700 REM DRAW SHAPE
705 REM
710 FOR J=1 TO NL
720 DS=1
730 P=C(J)-A(J)
740 Q=D(J)-B(J)
750 R=SQR(P*P+Q*Q)
760 LX=P/R
770 LY=Q/R
780 FOR I=0 TO R STEP DS
790 X=6*(A(J)+I*LX-40)
795 REM
800 Y=950-6*(B(J)+I*LY)
802 IF X<0 OR Y<0 THEN 820
804 IF X>1023 OR Y>950 THEN 820
805 REM
```

```
810 POINT 3,X,Y
820 NEXT I
830 NEXT J
840 GOTO 280:REM DO AGAIN
900 REM END PROGRAM
910 COLOR 1,3,6,0
920 GRAPHIC 0
930 END
995 REM
1000 REM INPUT DATA
1005 REM
1010 GET A$:IF A$="" THEN 1010
1020 IF (ASC(A$)<48 OR ASC(A$)>57) AND A$<>"." THEN 1060
1030 Z$=Z$+A$
1035 CHAR 19,T,A$
1040 T=T+1
1050 GOTO 1010
1060 Z=VAL(Z$)
1070 RETURN
1095 REM
1100 REM DRAW BORDER AROUND SCREEN
1105 REM
1110 POINT 3,0,0
1120 DRAW 3 TO 1023,0
1130 DRAW 3 TO 1023,950
1140 DRAW 3 TO 0,950
1150 DRAW 3 TO 0,0
1160 RETURN

READY.
```

ROTATING and MOVING

ROTATE

DESCRIPTION

In this section we introduce the concept of a transformation matrix. A transformation matrix is essentially a set of equations which are applied to a coordinate point in order to move it to the required position. I shall not endeavour to derive these equations (there are many excellent text books on the subject) simply show how they can be used to produce the required effects. The rotational transformation matrix consists of four equations and these are calculated in lines 250 to 280. Lines 290-300 use the values from this matrix to calculate the new coordinates of the point.

Rotation requires the movement of a point in a circle around a fixed axis on the screen. By making the point the end coordinate of a line, a line or a shape can be rotated around this axis. The axis of rotation can lie anywhere on the screen, it may even lie on the same coordinates as the point to be rotated. In this program you will notice that the small cross is being rotated in a clockwise direction around an axis thereby describing a circle, note that the point erase — lines 310 to 340 — was removed to produce the diagram. Counterclockwise rotation can be produced by using a negative angle of rotation.

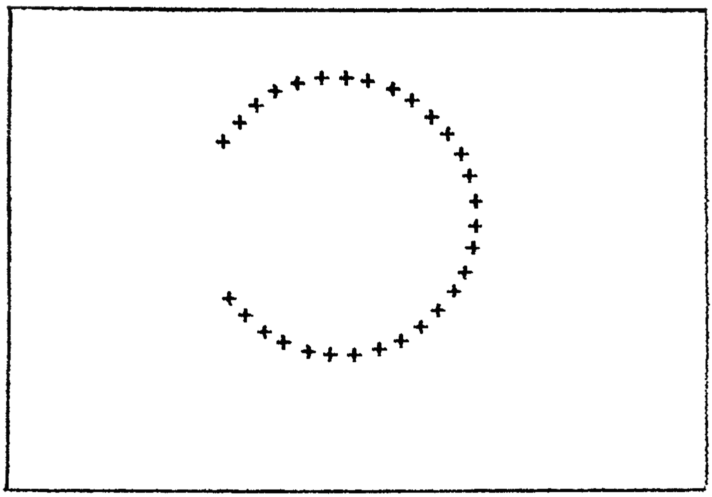
RUNNING THE PROGRAM

The program requires the input of five parameters. These five are the X and Y coordinates of the centre of rotation, the X and Y coordinates of the point to be rotated and the angle of rotation. The angle of rotation is in degrees and is the angle between two lines drawn from the centre of rotation to the 0 degree or three o'clock position and from the centre to the new point position. It should be noted that the FOR NEXT loop in lines 235 and 410 is inserted to generate a sequence of 360 rotational plot points; these should be removed to plot a single rotation.

PROGRAM STRUCTURE

35	
60-70	set colours
90	draw border around screen using subroutine at 500
110-119	input coordinates for centre of rotation
120-129	input coordinates for point to be plotted
130-136	input angle of rotation

210	convert rotation angle from degrees to radians
215-220	initialise variables
225	plot point at centre of rotation
230	set start angle at 0
235	loop to plot 20 consecutive rotations
240	add angle of rotation to start angle
250-280	calculate rotational transform matrix
290-300	calculate new coordinate point position
310-340	erase previous rotated point position
350-380	plot new rotated point
410	loop to rotate again by the rotation angle
500-560	border drawing subroutine
600-660	input routine



```
1 REM ROTATE
2 REM ************************
3 REM
10 REM THIS PROGRAM ROTATES A POINT AROUND
20 REM A CENTRAL POINT ON THE SCREEN
30 REM
35 DIM M(2,2)
40 REM
50 REM SET COLOURS
55 REM
60 GRAPHIC 2
70 COLOR 3,3,0,3
75 REM
80 REM DRAW BORDER
85 REM
90 GOSUB 500
95 REM
100 REM INPUT PARAMETERS
105 REM
110 REM COORDINATES OF CENTRE OF ROTATION
111 Z$="":T=5
112 CHAR 19,2,"?"
113 GOSUB 600
114 XC=Z:Z$=""
115 CHAR 19,T,",":T=T+1
116 GOSUB 600
117 YC=Z:FOR I=1 TO 500:NEXT I
118 IF XC=0 OR YC=0 THEN 450
119 CHAR 19,2,"                    "
120 REM COORDINATES OF POINT TO BE ROTATED
121 Z$="":T=5
122 CHAR 19,2,"?"
123 GOSUB 600
124 XP=Z:Z$=""
125 CHAR 19,T,",":T=T+1
126 GOSUB 600
127 YP=Z
128 FOR I=1 TO 500:NEXT I
129 CHAR 19,2,"                    "
130 REM ANGLE OF ROTATION
131 Z$="":T=5
132 CHAR 19,2,"?"
133 GOSUB 600
134 AR=Z
135 FOR I=1 TO 500:NEXT I
136 CHAR 19,2,"                    "
195 REM
200 REM ROTATE POINT
205 REM
210 AR=AR*π/180
215 XR=XP:YR=YP
```

```
217 XO=XR:YO=YR
220 XP=-(XC-XP):YP=-(YC-YP)
225 POINT 3,XC,YC
230 R=0
235 FOR Q=1 TO 20
240 R=R+AR
250 M(1,1)=COS(R)
260 M(1,2)=SIN(R)
270 M(2,1)=-SIN(R)
280 M(2,2)=COS(R)
290 X=XC+0.7*(XP*M(1,1)+YP*M(2,1))
300 Y=YC+(XP*M(1,2)+YP*M(2,2))
310 POINT 4,XO-12,YO
320 DRAW 4 TO XO+12,YO
330 POINT 4,XO,YO-12
340 DRAW 4 TO XO,YO+12
350 POINT 3,XR-12,YR
360 DRAW 3 TO XR+12,YR
370 POINT 3,XR,YR-12
380 DRAW 3 TO XR,YR+12
390 XO=XR:YO=YR
400 XR=X:YR=Y
410 NEXT Q
420 GOTO 100
450 COLOR 1,3,6,0
460 GRAPHIC 0
470 END
495 REM
500 REM DRAW BORDER
505 REM
510 POINT 3,0,0
520 DRAW 3 TO 0,950
530 DRAW 3 TO 1023,950
540 DRAW 3 TO 1023,0
550 DRAW 3 TO 0,0
560 RETURN
595 REM
600 REM INPUT DATA
605 REM
610 GET A$:IF A$="" THEN 610
620 IF (ASC(A$)<48 OR ASC(A$)>57) AND A$<>"." THEN 650
630 CHAR 19,T,A$:T=T+1
640 Z$=Z$+A$:GOTO 610
650 Z=VAL(Z$)
660 RETURN

READY.
```

ROTATE 2

DESCRIPTION

In the same way that the program ROTATE rotated a point around a fixed axis on the screen we can also rotate a line about a fixed axis. This is not difficult since one is simply rotating two points — the two end coordinates of the line. It should be noted that in this program the line start and end coordinates are both input as relative coordinates. A relative coordinate means that the coordinate is not the normal screen coordinate but a value which is relative to the coordinate of the axis point. If the axis is set at the absolute screen coordinates of X = 100 and Y = 80 then to have the start of the line at the absolute screen coordinates of X = 150 and Y = 100 gives us a relative coordinate value of X = 50 and Y = 20. From this we can see that the relative coordinates are obtained by this calculation:

coordinate of point — axis coordinate

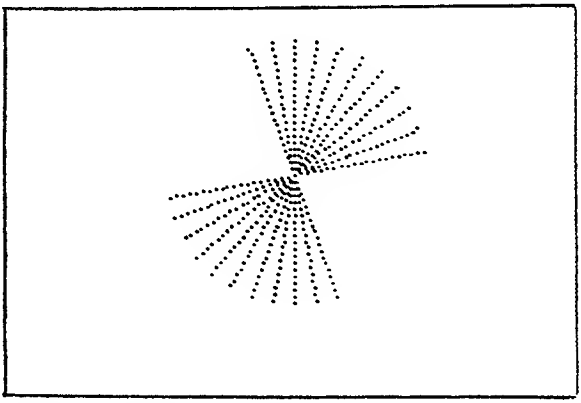
RUNNING THE PROGRAM

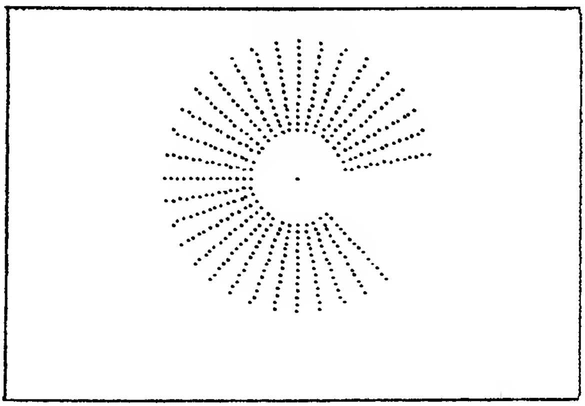
The program requires the input of seven parameters, starting with the X and Y coordinates of the central axis around which the line is rotated. This is followed by the X and Y coordinates of the start of the line and then the X and Y coordinates of the end of the line, all four values being relative coordinates with respect to the centre of rotation. The last parameter value is the angle of rotation, this is in degrees and is the angle between two lines drawn from the centre of rotation to the original dot position and from the centre to the new dot position. Note that the FOR NEXT loop in lines 235 and 500 has been inserted to generate a sequence of 20 rotations of the increment angle. These should be removed to plot a single rotation.

PROGRAM STRUCTURE

40	
60-70	set colours
90	draw border around screen using subroutine at 700
110-119	input coordinates for centre of rotation
120-129	input relative coordinates for start of line
130-139	input relative coordinates for end of line
140-146	input angle of rotation
210	convert angle to radians
215	initialise variables

225	plot point at centre of rotation
230	set start angle at zero
	loop to plot 20 consecutive rotation increments
240	add angle of rotation to start angle
250-280	calculate rotational transform matrix
290-340	calculate new coordinate point positions
360-500	routine to draw line between the two end points
510	loop to next rotation increment
700-760	border drawing subroutine
800-860	input routine
900-930	end





```
1 REM ROTATE 2
2 REM ***********************
3 REM
10 REM THIS PROGRAM ROTATES A LINE AROUND
20 REM A CENTRAL POINT ON THE SCREEN
30 REM
40 DIM M(2,2)
45 REM
50 REM SET COLOURS
55 REM
60 GRAPHIC 2
70 COLOR 3,3,0,10
75 REM
80 REM DRAW BORDER AROUND SCREEN
85 REM
90 GOSUB 700
95 REM
100 REM INPUT PARAMETERS
110 REM COORDINATES OF CENTRE OF ROTATION
111 Z$="":T=5
112 CHAR 19,2,"?"
113 GOSUB 800
114 XC=Z:Z$=""
115 CHAR 19,T,",":T=T+1
116 GOSUB 800
117 YC=Z:FOR I=1 TO 500:NEXT I
118 IF XC=0 OR YC=0 THEN 900
119 CHAR 19,2,"                         "
120 REM RELATIVE LINE START COORDINATES
121 Z$="":T=5
122 CHAR 19,2,"?"
123 GOSUB 800
124 XP=Z:Z$=""
125 CHAR 19,T,",":T=T+1
126 GOSUB 800
127 YP=Z
128 FOR I=1 TO 500:NEXT I
129 CHAR 19,2,"                         "
130 REM RELATIVE LINE END COORDINATES
131 Z$="":T=5
132 CHAR 19,2,"?"
133 GOSUB 800
134 XQ=Z:Z$=""
135 CHAR 19,T,",":T=T+1
136 GOSUB 800
137 YQ=Z
138 FOR I=1 TO 500:NEXT I
139 CHAR 19,2,"                         "
140 REM ANGLE OF ROTATION
141 Z$="":T=5
142 CHAR 19,2,"?"
```

```
143 GOSUB 800
144 AR=Z
145 FOR I=1 TO 500:NEXT I
146 CHAR 19,2,"                    "
195 REM
200 REM ROTATE LINE
205 REM
210 AR=AR*π/180
215 XR=XP:YR=YP
225 POINT 3,XC,YC
230 R=0
235 FOR Z=1 TO 20
240 R=R+AR
250 M(1,1)=COS(R)
260 M(1,2)=SIN(R)
270 M(2,1)=-SIN(R)
280 M(2,2)=COS(R)
290 X=XC+XP*M(1,1)+YP*M(2,1)
300 Y=YC+XP*M(1,2)+YP*M(2,2)
310 XB=X:YB=Y
320 X=XC+XQ*M(1,1)+YQ*M(2,1)
330 Y=YC+XQ*M(1,2)+YQ*M(2,2)
340 XE=X:YE=Y
345 REM
350 REM DRAW LINE
355 REM
360 DS=18
370 P=XE-XB
380 Q=YE-YB
390 RL=SQR(P*P+Q*Q)
400 LX=P/RL
410 LY=Q/RL
420 FOR I=0 TO RL STEP DS
430 X=XB+.7*(I*LX)
440 Y=YB+I*LY
445 IF X<0 OR Y<0 OR Y>950 THEN 460
450 POINT 3,X,Y
460 NEXT I
500 NEXT Z
510 GOTO 100
695 REM
700 REM DRAW BORDER
705 REM
710 POINT 3,0,0
720 DRAW 3 TO 0,950
730 DRAW 3 TO 1023,950
740 DRAW 3 TO 1023,0
750 DRAW 3 TO 0,0
760 RETURN
795 REM
```

```
800 REM INPUT DATA
805 REM
810 GET A$:IF A$="" THEN 810
820 IF (ASC(A$)<48 OR ASC(A$)>57) AND A$<>"." THEN 850
830 CHAR 19,T,A$:T=T+1
840 Z$=Z$+A$:GOTO 810
850 Z=VAL(Z$)
860 RETURN
900 REM END PROGRAM
910 COLOR 1,3,6,0
920 GRAPHIC 0
930 END

READY.
```

ROTATE 3

DESCRIPTION

In the same way that the program ROTATE 2 rotated a line around a fixed axis on the screen we can also rotate a shape about a fixed axis. This is not difficult since one is simply rotating a set of lines, each line being specified by the two end coordinates of the line. The data for the shape is stored in a shape table, this is stored in one of three arrays. The other two arrays are used to store the data for the rotated shape and the previous rotation — this is required by the routine which erases the previous rotation. The data is stored as the beginning X and Y coordinate of a line followed by the end X and Y coordinates of the same line, these four values are then repeated for each line in the shape. In this program the shape data is obtained from a set of data statements — lines 710 to 740. The set of displays which accompany this program show how by varying the centre of rotation the shape is rotated in different ways, depending on whether the rotational centre lies within the shape, directly on a line of axis through the shape or to one side of the shape; also shown is that the lines used to draw the shape can have a variable dot spacing.

RUNNING THE PROGRAM

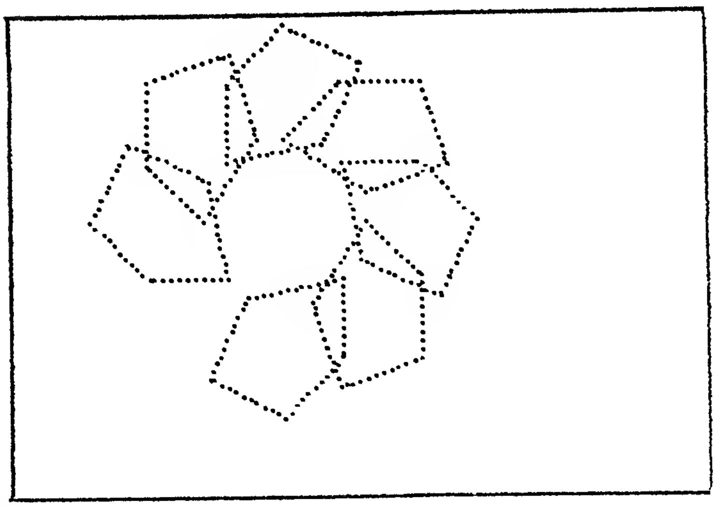
All the parameters required by the program are stored directly within the program. The X and Y coordinates of the central axis around which the shape is rotated are stored as the variables xc and yc in line 250. The number of lines in the shape is stored as variable nl in line 240. The X and Y coordinates of the start and end of each line are stored as data statements in lines 710 to 740. The last parameter value is the angle of rotation, this is in degrees and is stored as the variable ar in line 296.

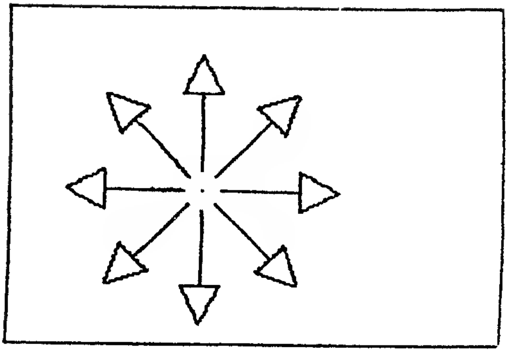
Note: that the FOR NEXT loop in lines 300 and 620 has been inserted to generate a sequence of fifty rotations of the increment angle. These should be removed to plot a single rotation. When plotting shapes with more than 20 lines then the size of the shape data arrays should be increased accordingly.

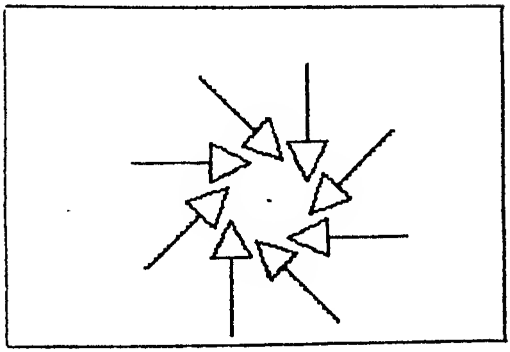
PROGRAM STRUCTURE

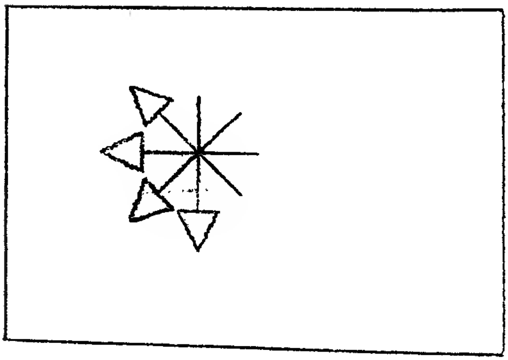
60-70	set colours
90	draw border around screen using subroutine at 900

110	set up array for rotation matrix
120-150	matrix for original data shape
160-190	matrix for erased shape data
210-225	matrix for displayed shape data
234-296	initialise variables and constants
240	number of lines in shape
255	plot point at centre of rotation
260-290	load coordinate data into original shape matrix
296	set start angle to zero
300	loop to plot 50 consecutive rotation increments
310	add angle of rotation to start angle
320-350	calculate rotational transform matrix
370-500	calculate new coordinate point positions
520	jump routine to erase lines
540	jump to routine to draw lines
560-610	put displayed shape data into erased shape matrix
620	loop to next rotation increment
710-740	shape table data
900-960	border drawing subroutine
1000-1140	subroutine to draw shape
2000-2140	subroutine to erase shape









```
1 REM ROTATE 3
2 REM ***********************
3 REM
10 REM PROGRAM TO ROTATE A 2D OBJECT ABOUT
20 REM A POINT ON THE SCREEN
30 REM
40 REM
50 REM SET COLOURS
55 REM
60 GRAPHIC 2
70 COLOR 3,3,0,3
75 REM
80 REM DRAW BORDER
85 REM
90 GOSUB 900
95 REM
100 REM ARRAYS FOR DATA TRANSFORMATION
105 REM
106 REM ROTATION MATRIX
107 REM
110 DIM M(2,2)
115 REM
116 REM ORIGINAL SHAPE DATA
117 REM
120 DIM X(20)
130 DIM Y(20)
135 REM
140 DIM U(20)
150 DIM V(20)
155 REM
156 REM ERASED SHAPE DATA
157 REM
160 DIM W(20)
170 DIM Z(20)
175 REM
180 DIM S(20)
190 DIM T(20)
200 REM
205 REM DISPLAYED SHAPE DATA
206 REM
210 DIM O(20)
215 DIM P(20)
220 DIM Q(20)
225 DIM R(20)
230 REM
234 REM SET UP CONSTANTS AND DATA FROM DATA TABLES
235 REM
240 NL=4
250 XC=450:YC=512
255 POINT 3,XC,YC
260 FOR N=1 TO NL
```

```
270 READ X(N),Y(N),U(N),V(N):S(N)=U(N):T(N)=V(N)
280 W(N)=X(N):Z(N)=Y(N)
290 NEXT N
296 AR=45:R=0
297 REM
298 AR=AR*π/180
299 REM
300 FOR A=1 TO 50
305 REM
310 R=R+AR
315 REM
316 REM SET UP ROTATION MATRIX
317 REM
320 M(1,1)=COS(R)
330 M(1,2)=SIN(R)
340 M(2,1)=-SIN(R)
350 M(2,2)=COS(R)
360 REM
365.REM ROTATE SHAPE AR DEGREES
366 REM
370 FOR N=1 TO NL
380 P=-(XC-X(N))
390 Q=-(YC-Y(N))
400 X=XC+.7*(P*M(1,1)+Q*M(2,1))
410 Y=YC+P*M(1,2)+Q*M(2,2)
420 O(N)=X
430 P(N)=Y
440 P=-(XC-U(N))
450 Q=-(YC-V(N))
460 X=XC+.7*(P*M(1,1)+Q*M(2,1))
470 Y=YC+P*M(1,2)+Q*M(2,2)
480 Q(N)=X
490 R(N)=Y
500 NEXT N
510 REM
520 GOSUB 2000
530 REM
540 GOSUB 1000
550 REM
560 FOR N=1 TO NL
570 W(N)=O(N)
580 Z(N)=P(N)
590 S(N)=Q(N)
600 T(N)=R(N)
610 NEXT N
620 NEXT A
630 GET A$:IF A$="" THEN 630
640 COLOR 1,3,6,0
650 GRAPHIC 0
660 END
695 REM
```

```
700 REM SHAPE DATA
705 REM
710 DATA 512,450,512,300
720 DATA 512,200,570,300
730 DATA 570,300,455,300
740 DATA 455,300,512,200
895 REM
900 REM DRAW BORDER
905 REM
910 POINT 3,0,0
920 DRAW 3 TO 0,1023
930 DRAW 3 TO 1023,1023
940 DRAW 3 TO 1023,0
950 DRAW 3 TO 0,0
960 RETURN
995 REM
1000 REM DRAW SHAPE
1005 REM
1010 FOR N=1 TO NL
1020 DS=6
1030 P=Q(N)-O(N)
1040 Q=R(N)-P(N)
1050 RL=SQR(P*P+Q*Q)
1060 LX=P/RL
1070 LY=Q/RL
1080 FOR I=0 TO RL STEP DS
1090 X=(O(N)+I*LX)
1100 Y=P(N)+I*LY
1105 IF X<0 OR Y<0 THEN 1120
1110 POINT 3,X,Y
1120 NEXT I
1130 NEXT N
1140 RETURN
1995 REM
2000 REM ERASE SHAPE
2005 REM
2010 FOR N=1 TO NL
2020 DS=6
2030 P=S(N)-W(N)
2040 Q=T(N)-Z(N)
2050 RL=SQR(P*P+Q*Q)
2060 LX=P/RL
2070 LY=Q/RL
2080 FOR I=0 TO RL STEP DS
2090 X=(W(N)+I*LX)
2100 Y=Z(N)+I*LY
2105 IF X<0 OR Y<0 THEN 2020
2110 POINT 4,X,Y
2120 NEXT I
2130 NEXT N
2140 RETURN
```

MOVE

DESCRIPTION

The application of the transformation matrix can be expanded to cover all manipulation of a shape, not just rotation but also movement (known as translation) and scaling. The primary purpose of this program is to show how a shape can be moved about the screen, but it also embodies the capability of scaling and rotation. The transformation matrix consists of six quotations. These equations are stored in lines 3000 to 3100. Notice that equations 1 to 4 consist of the rotational transform equation multiplied by a scaling factor, equations 5 and 6 do the movement by adding an offset to the shape position. The program can display any two dimensional shape. This shape can be moved to any part of the screen, rotated through 360 degrees and stretched in either X or Y axis or both.

RUNNING THE PROGRAM

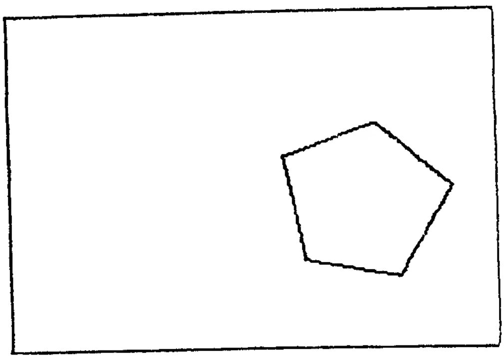
There are no input parameter values since they are all within the program as LET statements. There are five parameter values which control the movement, rotation or scaling of the shape; these are set in lines 120 to 160. Lines 120 and 130 contain the X and Y scaling factors — full size =1, half size =.5 etc. The rotational angle of the shape is stored as the variable rx in line 140, note that since this angle must be in radians it is multiplied by 3.14159/180. The movement of the shape in the X and Y axis is stored in lines 150 and 160, and is the number of pixels in either direction from the original coordinates stored in the shape table.

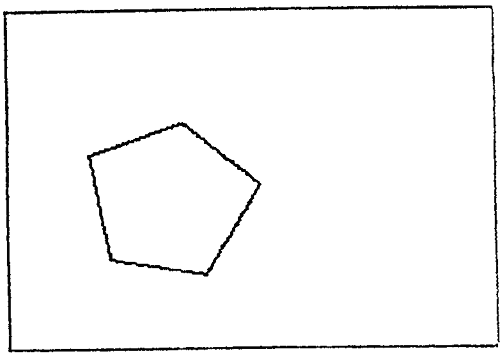
The object shape is stored in a shape table. This table consists simply of the X and Y coordinates of the end of each line comprising the shape. It should be noted that there are one more pair of coordinates than there are lines in the shape, the number of lines in the shape is stored as the variable np as the first value in the data table. The data table is stored as data statements in lines 1110 to 1130. Try designing your own shapes using graph paper and then entering the new values into the data statements.

PROGRAM STRUCTURE

60-70	set colours
90	draw border around screen using subroutine at 400
110	set up transform matrix array

120-130	X and Y scaling factors
140	angle of shape rotation in radians
150-160	X and Y axis movement of shape from initial position
210-260	main program execution loop
400-460	border drawing subroutine
1000-1050	load shape data into arrays — arrays X and Y contain the original shape data — arrays U and V contain the transformed shape data
1110-1130	data statements containing shape data — line 110 contains the number of lines in the shape
2000-2080	find the centre of the shape
3000-3100	perform transformation matrix calculations
4000-4070	performs the transformation on each coordinate point within the shape table
5000-5220	draws the shape using the transformed data in the arrays U and V, note lines 5120 and 5130 check that the shape does not fall outside the screen area





```
1 REM MOVE
2 REM ************************
3 REM
10 REM THIS PROGRAM USES MATRIX TRANSFORMATION TO
20 REM MOVE,ROTATE, OR SCALE A TWO DIMENSIONAL SHAPE
30 REM
40 REM
50 REM SET COLOURS
60 GRAPHIC 2
70 COLOR 3,3,0,10
75 REM
80 REM DRAW BORDER
90 GOSUB 400
95 REM
100 REM SET UP CONSTANTS, VARIABLES, AND ARRAYS
105 REM
110 DIM A(3,3)
120 SX=1
130 SY=1
140 RX=80*π/180
150 TX=-50
160 TY=2
190 REM
200 REM MAIN PROGRAM LOOP
205 REM
210 GOSUB 1000
220 GOSUB 2000
230 GOSUB 3000
240 GOSUB 4000
250 GOSUB 5000
260 GET A$:IF A$="" THEN 260
270 COLOR 1,3,6,0
280 GRAPHIC 0
290 END
395 REM
400 REM DRAW BORDER
405 REM
410 POINT 3,0,0
420 DRAW 3 TO 0,1023
430 DRAW 3 TO 1023,1023
440 DRAW 3 TO 1023,0
450 DRAW 3 TO 0,0
460 RETURN
995 REM
1000 REM INITIALISE SHAPE
1005 REM
1010 READ NP
1020 DIM X(NP+1),Y(NP+1),U(NP+1),V(NP+1)
1030 FOR I=1 TO NP+1
1040 READ X(I),Y(I)
1050 NEXT I
```

```
1090 REM
1100 REM SHAPE DATA
1105 REM
1110 DATA 5
1120 DATA 100,100,150,120,175,75
1130 DATA 150,30,100,50,100,100
1200 RETURN
1995 REM
2000 REM FIND CENTRE OF SHAPE
2005 REM
2010 CX=0:CY=0
2020 FOR C=1 TO NP
2030 CX=CX+X(C)
2040 CY=CY+Y(C)
2050 NEXT C
2060 CX=CX/NP
2070 CY=CY/NP
2080 RETURN
2995 REM
3000 REM SET TRANSFORMATION MATRIX
3005 REM
3010 A(1,1)=SX*COS(RZ)
3020 A(1,2)=SX*SIN(RZ)
3030 REM
3040 A(2,1)=SY*(-SIN(RZ))
3050 A(2,2)=SY*COS(RZ)
3060 REM
3070 A(3,1)=TX
3080 A(3,2)=TY
3090 REM
3100 RETURN
3995 REM
4000 REM DO TRANSFORMATION
4005 REM
4010 FOR Q=1 TO NP+1
4020 XT=X(Q)-CX
4030 YT=Y(Q)-CY
4040 U(Q)=CX+(XT*A(1,1)+YT*A(2,1)+A(3,1))
4050 V(Q)=CY+(XT*A(1,2)+YT*A(2,2)+A(3,2))
4060 NEXT Q
4070 RETURN
4995 REM
5000 REM DRAW SHAPE
5005 REM
5010 FOR Q=1 TO NP
5020 XB=U(Q):YB=V(Q)
5030 XE=U(Q+1):YE=V(Q+1)
5040 P=XE-XB
5050 O=YE-YB
5060 R=SQR(P*P+O*O)
5070 LX=P/R
```

```
5080 LY=O/R
5090 FOR I=0 TO R STEP 1
5100 X=6*.7*(XB+I*LX)
5110 Y=1023-6*(YB+I*LY)
5120 IF X<0 OR Y<0 THEN 5200
5130 IF Y>1023 OR X>1023 THEN 5200
5190 POINT 3,X,Y
5200 NEXT I
5210 NEXT Q
5220 RETURN

READY.
```

3D DISPLAYS

THREE DIMENSIONAL SHAPE 1

DESCRIPTION

The application of the transformation matrix can be expanded further to cover the generation of three dimensional shapes — it should be noted that they are displayed two dimensionally but optically appear to represent three dimensional objects. To do this simply requires the addition of an extra axis — the Z axis — to the X and Y axis used in a two dimensional transformation matrix. The transformation matrix consists of sixteen equations, they are stored in lines 3000 to 3190. I shall not attempt to explain the mathematics, for those interested I would suggest one of the text books on the subject — 'Principles of Interactive Graphics' by Newman and Sproul.

RUNNING THE PROGRAM

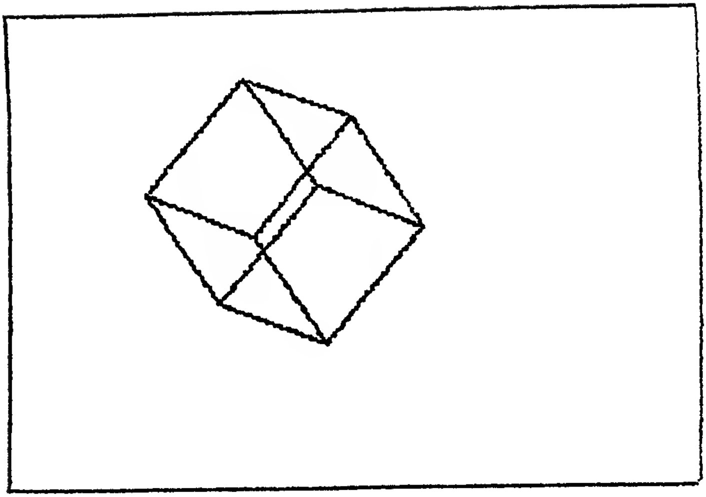
There are no input parameter values since they are all within the program as LET statements. There are nine parameter values which control the movement, rotation or scaling of the shape, these are set in lines 120 to 200. Lines 120 and 140 contain the X,Y and Z scaling factors — full size = 1, half size = .5 etc. The rotational angle of the shape in either one of the three axis are stored in lines 180 to 200, note that since these angles must be in radians they are multiplied by 3.14159/180. The movement of the shape in the X,Y and Z axis is stored in lines 150 to 170, and is the number of pixels in either direction from the original coordinates stored in the shape table.

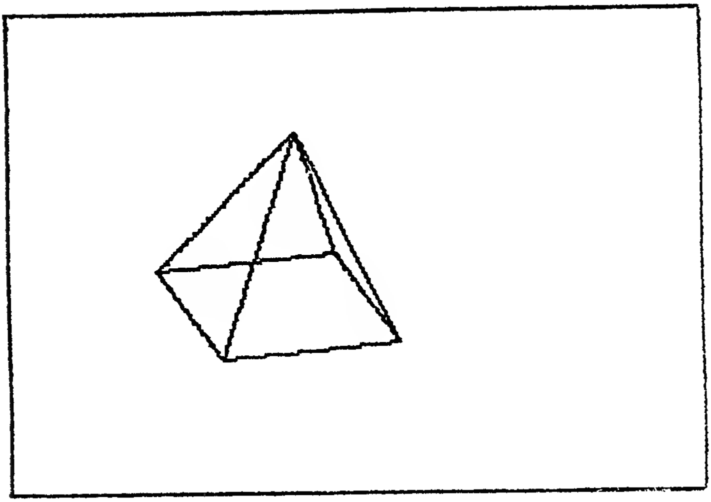
The object shape is stored in a shape table. This table consists of two parts the first is simply of the X,Y and Z coordinates, of each cordner coordinate comprising the shape. The second part is a table of connections of pairs of points between which a line should be drawn. The number of edges in the shape is stored as the variable 'ne' and the number of coordinate points between which the edges are connected is stored as variable 'np'. The coordinate table is stored as data statements in lines 1210 to 1220, and the connection table in lines 1310 to 1330.
Note: all programs in this section require 16K RAM expansion.

PROGRAM STRUCTURE

50-60	set colours
80	draw border around screen using subroutine at 900

100-110	set up transform matrix arrays
120-140	X,Y and Z scaling factors
150-170	X,Y and Z axis movement of shape from initial position
180-200	angle of X,Y and Z axis rotation in radians
410-450	main program execution loop
900-960	border drawing subroutine
1000-1050	load shape data into arrays — array S contains the coordinate table of the original shape — array E contains the line connection data — array M contains the transformed coordinate data
1100-1170	read in the shape data
1200-1220	data statements containing coordinate shape data as X,Y and Z for each corner point, note that the first three values comprise the coordinates for point 1, the second three for point 2 etc
1300-1330	data statements containing line connection data
2000-2240	draw the shape
3000-3160	perform transformation matrix calculations
3200-3350	set up scaling and translation matrix
4000-4080	performs the transformation on each coordinate point within the shape table
5000-5900	find centre of shape





```
1 REM 3D DRAWING 1
2 REM ***********************
3 REM
10 REM A THREE DIMENSIONAL SHAPE IS DRAWN BY THIS PROGRAM
20 REM THE ROTATION POSITION AND SCALE OF THE OBJECT
30 REM CAN BE CHANGED TO GIVE DIFFERENT VIEWING ANGLES.
35 REM
40 REM SET COLOURS
50 GRAPHIC 2
60 COLOR 3,3,0,10
65 REM
70 REM DRAW BORDER AROUND SCREEN
75 REM
80 GOSUB 900
85 REM
90 REM SET UP CONSTANTS, VARIABLES, AND ARRAYS
95 REM
100 DIM A(4,4)
110 DIM B(4,4)
120 SX=.3
130 SY=.3
140 SZ=.3
150 TX=1
160 TY=1
170 TZ=1
180 RX=40*π/180
190 RY=20*π/180
200 RZ=50*π/180
400 REM MAIN PROGRAM LOOP
410 GOSUB 1000
420 GOSUB 5000
430 GOSUB 3000
440 GOSUB 4000
450 GOSUB 2000
500 GET A$:IF A$="" THEN 500
510 COLOR 1,3,6,0
520 GRAPHIC 0
530 END
900 REM BORDER DRAWING SUBROUTINE
905 REM
910 POINT 3,0,0
920 DRAW 3 TO 0,1023
930 DRAW 3 TO 1023,1023
940 DRAW 3 TO 1023,0
950 DRAW 3 TO 0,0
960 RETURN
995 REM
1000 REM INITIALISE SHAPE
1005 REM
1010 NP=8
1020 NE=12
```

```
1030 REM
1040 DIM S(3,NP)
1050 DIM E(NE,2)
1060 DIM M(3,NP)
1100 REM
1110 FOR N=1 TO NP
1120 READ S(1,N),S(2,N),S(3,N)
1130 NEXT N
1140 FOR K=1 TO NE
1150 READ E(K,1),E(K,2)
1170 NEXT K
1195 REM
1200 REM X,Y,Z POINT COORDINATES
1210 DATA 0,0,200,200,0,200,200,0,0,0,0,0
1220 DATA 0,200,200,200,200,200,200,200,0,0,200,0
1295 REM
1300 REM CONNECTION DATA
1305 REM
1310 DATA 1,2,2,3,3,4,4,1
1320 DATA 5,1,2,6,4,8,7,3
1330 DATA 6,5,5,8,8,7,7,6
1900 RETURN
1995 REM
2000 REM DRAW SHAPE
2005 REM
2020 FOR K=1 TO NE
2030 V1=E(K,1)
2040 V2=E(K,2)
2045 IF V1=0 THEN 2240
2050 XB=M(1,V1)
2060 YB=M(2,V1)
2070 XE=M(1,V2)
2080 YE=M(2,V2)
2090 DS=1
2100 P=XE-XB
2110 Q=YE-YB
2120 R=SQR(P*P+Q*Q)
2130 LX=P/R
2140 LY=Q/R
2150 FOR I=0 TO R STEP DS
2160 X=6*0.7*(XB+I*LX)
2170 Y=1023-6*(YB+I*LY)
2180 IF X<0 OR Y<0 THEN 2230
2190 IF X>1023 OR Y>1023 THEN 2230
2220 POINT 3,X,Y
2230 NEXT I
2240 NEXT K
2900 RETURN
2995 REM
3000 REM SET TRANSFORMATION MATRIX
3005 REM
```

```
3010 A(1,1)=COS(RY)*COS(RZ)
3020 A(1,2)=COS(RY)*SIN(RZ)
3030 A(1,3)=-SIN(RY)
3040 A(1,4)=0
3050 A(2,1)=COS(RX)*(-SIN(RZ))+SIN(RX)*SIN(RY)*COS(RZ)
3060 A(2,2)=COS(RX)*COS(RZ)+SIN(RX)*SIN(RY)*SIN(RZ)
3070 A(2,3)=SIN(RX)*COS(RY)
3080 A(2,4)=0
3090 A(3,1)=(-SIN(RX))*(-SIN(RZ))+COS(RX)*SIN(RY)*COS(RZ)
3100 A(3,2)=-SIN(RX)*COS(RZ)+COS(RZ)*SIN(RY)*SIN(RZ)
3110 A(3,3)=COS(RX)*COS(RY)
3120 A(3,4)=0
3130 A(4,1)=0
3140 A(4,2)=0
3150 A(4,3)=0
3160 A(4,4)=1
3195 REM
3200 REM SET UP SCALING AND TRANSLATION MATRIX
3205 REM
3210 B(1,1)=SX*A(1,1)
3220 B(1,2)=SX*A(1,2)
3230 B(1,3)=SX*A(1,3)
3240 REM
3250 B(2,1)=SY*A(2,1)
3260 B(2,2)=SY*A(2,2)
3270 B(2,3)=SY*A(2,3)
3280 REM
3290 B(3,1)=SZ*A(3,1)
3300 B(3,2)=SZ*A(3,2)
3310 B(3,3)=SZ*A(3,3)
3320 REM
3330 B(4,1)=TX
3340 B(4,2)=TY
3350 B(4,3)=TZ
3900 RETURN
3995 REM
4000 REM PERFORM TRANSLATION
4005 REM
4010 FOR Q=1 TO NP
4015 REM
4020 XT=S(1,Q)-XC
4030 YT=S(2,Q)-YC
4040 ZT=S(3,Q)-ZC
4045 REM
4050 M(1,Q)=XC+(XT*B(1,1)+YT*B(2,1)+ZT*B(3,1)+B(4,1))
4060 M(2,Q)=YC+(XT*B(1,2)+YT*B(2,2)+ZT*B(3,2)+B(4,2))
4070 M(3,Q)=ZC+(XT*B(1,3)+YT*B(2,3)+ZT*B(3,3)+B(4,3))
4080 NEXT Q
4900 RETURN
4995 REM
5000 REM FIND CENTROID
```

```
5005 REM
5010 P=0:Q=0:R=0
5020 FOR I=1 TO NP
5030 P=P+S(1,I)
5040 Q=Q+S(2,I)
5050 R=R+S(3,I)
5060 NEXT I
5070 XC=P/NP
5080 YC=Q/NP
5090 ZC=R/NP
5900 RETURN

READY.
```

THREE DIMENSIONAL SHAPE 2

DESCRIPTION

This program is identical to the program THREE DIMENSIONAL SHAPE 1 except that an additional subroutine has been added to remove hidden lines. Hidden lines are those lines which lie out of sight of the viewer and are hidden behind the front surfaces. By removing these hidden lines the shape of the object becomes much clearer. The subroutine which checks for hidden lines is located between line numbers 6000 and 6140.

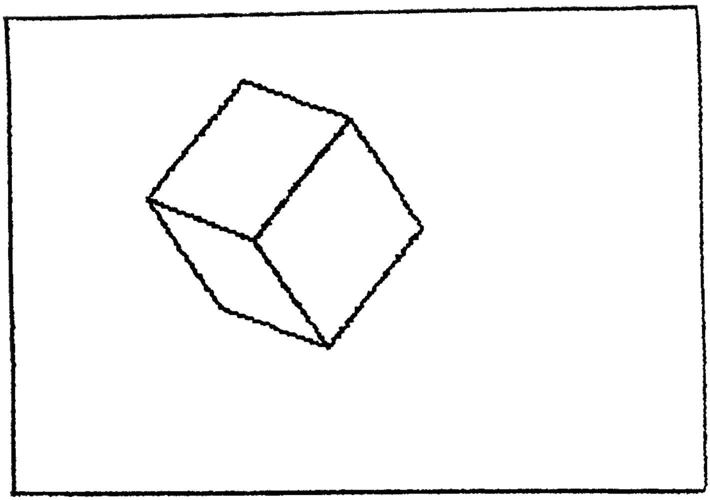
RUNNING THE PROGRAM

The parameters and data tables required by this program are the same as those used for the program THREE DIMENSIONAL SHAPE 1, consult this program for information. Note that the connection table now describes object faces rather than lines.

PROGRAM STRUCTURE

Lines 1 to 5995 are identical to THREE DIMENSIONAL SHAPE 1 consult for details.

6000-6140 subroutine to check for hidden surfaces



```
1 REM 3D DRAWING 2
2 REM ***********************
3 REM
10 REM A THREE DIMENSIONAL SHAPE IS DRAWN BY THIS
20 REM PROGRAM.THE ROTATION POSITION AND SCALE OF THE
30 REM OBJECT CAN BE CHANGED TO GIVE DIFFERENT VIEWING
35 REM ANGLES.THE PROGRAM INCORPORATES A ROUTINE TO
36 REM REMOVE HIDDEN LINES
37 REM
40 REM SET COLOURS
50 GRAPHIC 2
60 COLOR 3,3,0,10
65 REM
70 REM DRAW BORDER AROUND SCREEN
75 REM
80 GOSUB 900
85 REM
90 REM SET UP CONSTANTS, VARIABLES, AND ARRAYS
95 REM
100 DIM A(4,4)
110 DIM B(4,4)
115 DIM C(3)
117 DIM D(3)
120 SX=.3
130 SY=.3
140 SZ=.3
150 TX=1
160 TY=1
170 TZ=1
180 RX=40*π/180
190 RY=20*π/180
200 RZ=50*π/180
400 REM MAIN PROGRAM LOOP
410 GOSUB 1000
420 GOSUB 5000
430 GOSUB 3000
440 GOSUB 4000
450 GOSUB 6000
500 GET A$:IF A$="" THEN 500
510 COLOR 1,3,6,0
520 GRAPHIC 0
530 END
900 REM BORDER DRAWING SUBROUTINE
905 REM
910 POINT 3,0,0
920 DRAW 3 TO 0,1023
930 DRAW 3 TO 1023,1023
940 DRAW 3 TO 1023,0
950 DRAW 3 TO 0,0
960 RETURN
995 REM
```

```
1000 REM INITIALISE SHAPE
1005 REM
1010 NP=8
1020 NE=4
1025 NF=6
1030 REM
1040 DIM S(3,NP)
1050 DIM E(NF,NE,2)
1060 DIM M(3,NP)
1100 REM
1110 FOR N=1 TO NP
1120 READ S(1,N),S(2,N),S(3,N)
1130 NEXT N
1135 FOR F=1 TO NF
1140 FOR K=1 TO NE
1150 READ E(F,K,1),E(F,K,2)
1170 NEXT K
1180 NEXT F
1195 REM
1200 REM X,Y,Z POINT COORDINATES
1210 DATA 0,0,200,200,0,200,200,0,0,0,0,0
1220 DATA 0,200,200,200,200,200,200,200,0,0,200,0
1295 REM
1300 REM CONNECTION DATA
1305 REM
1310 DATA 1,2,2,3,3,4,4,1
1320 DATA 5,1,1,4,4,8,8,5
1330 DATA 6,5,5,8,8,7,7,6
1340 DATA 2,6,6,7,7,3,3,2
1350 DATA 1,5,5,6,6,2,2,1
1360 DATA 3,7,7,8,8,4,4,3
1900 RETURN
1995 REM
2000 REM DRAW SHAPE
2005 REM
2020 FOR K=1 TO NE
2030 V1=E(F,K,1)
2040 V2=E(F,K,2)
2045 IF V1=0 THEN 2240
2050 XB=M(1,V1)
2060 YB=M(2,V1)
2070 XE=M(1,V2)
2080 YE=M(2,V2)
2090 DS=1
2100 P=XE-XB
2110 Q=YE-YB
2120 R=SQR(P*P+Q*Q)
2130 LX=P/R
2140 LY=Q/R
2150 FOR I=0 TO R STEP DS
2160 X=6*0.7*(XB+I*LX)
```

```
2170 Y=1023-6*(YB+I*LY)
2180 IF X<0 OR Y<0 THEN 2230
2190 IF X>1023 OR Y>1023 THEN 2230
2220 POINT 3,X,Y
2230 NEXT I
2240 NEXT K
2900 RETURN
2995 REM
3000 REM SET TRANSFORMATION MATRIX
3005 REM
3010 A(1,1)=COS(RY)*COS(RZ)
3020 A(1,2)=COS(RY)*SIN(RZ)
3030 A(1,3)=-SIN(RY)
3040 A(1,4)=0
3050 A(2,1)=COS(RX)*(-SIN(RZ))+SIN(RX)*SIN(RY)*COS(RZ)
3060 A(2,2)=COS(RX)*COS(RZ)+SIN(RX)*SIN(RY)*SIN(RZ)
3070 A(2,3)=SIN(RX)*COS(RY)
3080 A(2,4)=0
3090 A(3,1)=(-SIN(RX))*(-SIN(RZ))+COS(RX)*SIN(RY)*COS(RZ)
3100 A(3,2)=-SIN(RX)*COS(RZ)+COS(RZ)*SIN(RY)*SIN(RZ)
3110 A(3,3)=COS(RX)*COS(RY)
3120 A(3,4)=0
3130 A(4,1)=0
3140 A(4,2)=0
3150 A(4,3)=0
3160 A(4,4)=1
3195 REM
3200 REM SET UP SCALING AND TRANSLATION MATRIX
3205 REM
3210 B(1,1)=SX*A(1,1)
3220 B(1,2)=SX*A(1,2)
3230 B(1,3)=SX*A(1,3)
3240 REM
3250 B(2,1)=SY*A(2,1)
3260 B(2,2)=SY*A(2,2)
3270 B(2,3)=SY*A(2,3)
3280 REM
3290 B(3,1)=SZ*A(3,1)
3300 B(3,2)=SZ*A(3,2)
3310 B(3,3)=SZ*A(3,3)
3320 REM
3330 B(4,1)=TX
3340 B(4,2)=TY
3350 B(4,3)=TZ
3900 RETURN
3995 REM
4000 REM PERFORM TRANSLATION
4005 REM
4010 FOR Q=1 TO NP
4015 REM
4020 XT=S(1,Q)-XC
```

```
4030 YT=S(2,Q)-YC
4040 ZT=S(3,Q)-ZC
4045 REM
4050 M(1,Q)=XC+(XT*B(1,1)+YT*B(2,1)+ZT*B(3,1)+B(4,1))
4060 M(2,Q)=YC+(XT*B(1,2)+YT*B(2,2)+ZT*B(3,2)+B(4,2))
4070 M(3,Q)=ZC+(XT*B(1,3)+YT*B(2,3)+ZT*B(3,3)+B(4,3))
4080 NEXT Q
4900 RETURN
4995 REM
5000 REM FIND CENTROID
5005 REM
5010 P=0:Q=0:R=0
5020 FOR I=1 TO NP
5030 P=P+S(1,I)
5040 Q=Q+S(2,I)
5050 R=R+S(3,I)
5060 NEXT I
5070 XC=P/NP
5080 YC=Q/NP
5090 ZC=R/NP
5900 RETURN
5995 REM
6000 REM HIDDEN SURFACE CHECK
6005 REM
6010 FOR F=1 TO NF
6020 FOR J=1 TO 3
6030 C(J)=M(J,E(F,1,2))-M(J,E(F,1,1))
6040 D(J)=M(J,E(F,2,1))-M(J,E(F,2,2))
6050 NEXT J
6060 P1=C(2)*D(3)-C(3)*D(2)
6070 P2=C(3)*D(1)-C(1)*D(3)
6080 P3=C(1)*D(2)-C(2)*D(1)
6090 Q1=1-M(1,E(F,1,2))
6100 Q2=1-M(2,E(F,1,2))
6110 Q3=500-M(3,E(F,1,2))
6120 W=P1*Q1+P2*Q2+P3*Q3
6130 IF W>=0 THEN GOSUB 2000
6140 NEXT F
6900 RETURN

READY.
```

THREE DIMENSIONAL SHAPE 3

DESCRIPTION

This program is identical to the program THREE DIMENSIONAL SHAPE 1 except that additional subroutines have been added to remove hidden lines, and to shade the faces of the displayed surfaces in respect of incident light coming from above in the Y axis. By shading the surfaces the viewer becomes fully aware of the shape of the three dimensional object as well as adding realism to the display.

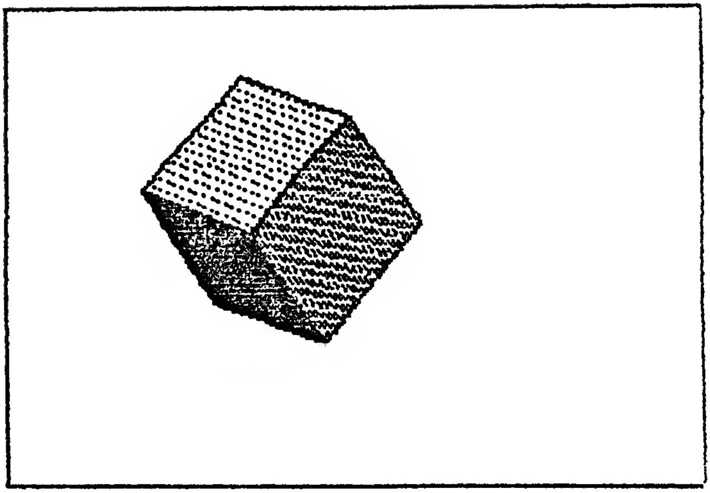
RUNNING THE PROGRAM

The parameters and data tables required by this program are the same as those used for the program THREE DIMENSIONAL SHAPE 2, consult this program for information.

PROGRAM STRUCTURE

Lines 1 to 5995 are identical to THREE DIMENSIONAL SHAPE 1 consult for details.

6000-6140	subroutine to check for hidden surfaces
7000-7330	shade the displayed surfaces



```
1 REM 3D DRAWING 3
2 REM ************************
3 REM
10 REM A THREE DIMENSIONAL SHAPE IS DRAWN BY THIS
20 REM PROGRAM.THE ROTATION POSITION AND SCALE OF THE
30 REM OBJECT CAN BE CHANGED TO GIVE DIFFERENT VIEWING
35 REM ANGLES.THE PROGRAM INCORPORATES A ROUTINE TO
36 REM REMOVE HIDDEN LINES.THE DISPLAYED FACES ARE
37 REM SHADED IN RESPECT OF INCIDENT LIGHT COMING
38 REM FROM ABOVE IN THE Y-AXIS.
39 REM
40 REM SET COLOURS
50 GRAPHIC 2
60 COLOR 3,3,0,10
65 REM
70 REM DRAW BORDER AROUND SCREEN
75 REM
80 GOSUB 900
85 REM
90 REM SET UP CONSTANTS, VARIABLES, AND ARRAYS
95 REM
100 DIM A(4,4)
110 DIM B(4,4)
115 DIM C(3)
117 DIM D(3)
120 SX=.3
130 SY=.3
140 SZ=.3
150 TX=1
160 TY=1
170 TZ=1
180 RX=40*π/180
190 RY=20*π/180
200 RZ=50*π/180
400 REM MAIN PROGRAM LOOP
410 GOSUB 1000
420 GOSUB 5000
430 GOSUB 3000
440 GOSUB 4000
450 GOSUB 6000
500 GET A$:IF A$="" THEN 500
510 COLOR 1,3,6,0
520 GRAPHIC 0
530 END
900 REM BORDER DRAWING SUBROUTINE
905 REM
910 POINT 3,0,0
920 DRAW 3 TO 0,1023
930 DRAW 3 TO 1023,1023
940 DRAW 3 TO 1023,0
950 DRAW 3 TO 0,0
```

```
960 RETURN
995 REM
1000 REM INITIALISE SHAPE
1005 REM
1010 NP=8
1020 NE=4
1025 NF=6
1030 REM
1040 DIM S(3,NP)
1050 DIM E(NF,NE,2)
1060 DIM M(3,NP)
1100 REM
1110 FOR N=1 TO NP
1120 READ S(1,N),S(2,N),S(3,N)
1130 NEXT N
1135 FOR F=1 TO NF
1140 FOR K=1 TO NE
1150 READ E(F,K,1),E(F,K,2)
1170 NEXT K
1180 NEXT F
1195 REM
1200 REM X,Y,Z POINT COORDINATES
1210 DATA 0,0,200,200,0,200,200,0,0,0,0,0
1220 DATA 0,200,200,200,200,200,200,200,0,0,200,0
1295 REM
1300 REM CONNECTION DATA
1305 REM
1310 DATA 1,2,2,3,3,4,4,1
1320 DATA 5,1,1,4,4,8,8,5
1330 DATA 6,5,5,8,8,7,7,6
1340 DATA 2,6,6,7,7,3,3,2
1350 DATA 1,5,5,6,6,2,2,1
1360 DATA 3,7,7,8,8,4,4,3
1900 RETURN
1995 REM
2000 REM DRAW SHAPE
2005 REM
2020 FOR K=1 TO NE
2030 V1=E(F,K,1)
2040 V2=E(F,K,2)
2045 IF V1=0 THEN 2240
2050 XB=M(1,V1)
2060 YB=M(2,V1)
2070 XE=M(1,V2)
2080 YE=M(2,V2)
2090 DS=1
2100 P=XE-XB
2110 Q=YE-YB
2120 R=SQR(P*P+Q*Q)
2130 LX=P/R
2140 LY=Q/R
```

```
2150 FOR I=0 TO R STEP DS
2160 X=6*0.7*(XB+I*LX)
2170 Y=1023-6*(YB+I*LY)
2180 IF X<0 OR Y<0 THEN 2230
2190 IF X>1023 OR Y>1023 THEN 2230
2220 POINT 3,X,Y
2230 NEXT I
2240 NEXT K
2300 GOSUB 7000
2900 RETURN
2995 REM
3000 REM SET TRANSFORMATION MATRIX
3005 REM
3010 A(1,1)=COS(RY)*COS(RZ)
3020 A(1,2)=COS(RY)*SIN(RZ)
3030 A(1,3)=-SIN(RY)
3040 A(1,4)=0
3050 A(2,1)=COS(RX)*(-SIN(RZ))+SIN(RX)*SIN(RY)*COS(RZ)
3060 A(2,2)=COS(RX)*COS(RZ)+SIN(RX)*SIN(RY)*SIN(RZ)
3070 A(2,3)=SIN(RX)*COS(RY)
3080 A(2,4)=0
3090 A(3,1)=(-SIN(RX))*(-SIN(RZ))+COS(RX)*SIN(RY)*COS(RZ)
3100 A(3,2)=-SIN(RX)*COS(RZ)+COS(RZ)*SIN(RY)*SIN(RZ)
3110 A(3,3)=COS(RX)*COS(RY)
3120 A(3,4)=0
3130 A(4,1)=0
3140 A(4,2)=0
3150 A(4,3)=0
3160 A(4,4)=1
3195 REM
3200 REM SET UP SCALING AND TRANSLATION MATRIX
3205 REM
3210 B(1,1)=SX*A(1,1)
3220 B(1,2)=SX*A(1,2)
3230 B(1,3)=SX*A(1,3)
3240 REM
3250 B(2,1)=SY*A(2,1)
3260 B(2,2)=SY*A(2,2)
3270 B(2,3)=SY*A(2,3)
3280 REM
3290 B(3,1)=SZ*A(3,1)
3300 B(3,2)=SZ*A(3,2)
3310 B(3,3)=SZ*A(3,3)
3320 REM
3330 B(4,1)=TX
3340 B(4,2)=TY
3350 B(4,3)=TZ
3900 RETURN
3995 REM
4000 REM PERFORM TRANSLATION
4005 REM
```

```
4010 FOR Q=1 TO NP
4015 REM
4020 XT=S(1,Q)-XC
4030 YT=S(2,Q)-YC
4040 ZT=S(3,Q)-ZC
4045 REM
4050 M(1,Q)=XC+(XT*B(1,1)+YT*B(2,1)+ZT*B(3,1)+B(4,1))
4060 M(2,Q)=YC+(XT*B(1,2)+YT*B(2,2)+ZT*B(3,2)+B(4,2))
4070 M(3,Q)=ZC+(XT*B(1,3)+YT*B(2,3)+ZT*B(3,3)+B(4,3))
4080 NEXT Q
4900 RETURN
4995 REM
5000 REM FIND CENTROID
5005 REM
5010 P=0:Q=0:R=0
5020 FOR I=1 TO NP
5030 P=P+S(1,I)
5040 Q=Q+S(2,I)
5050 R=R+S(3,I)
5060 NEXT I
5070 XC=P/NP
5080 YC=Q/NP
5090 ZC=R/NP
5900 RETURN
5995 REM
6000 REM HIDDEN SURFACE CHECK
6005 REM
6010 FOR F=1 TO NF
6020 FOR J=1 TO 3
6030 C(J)=M(J,E(F,1,2))-M(J,E(F,1,1))
6040 D(J)=M(J,E(F,2,1))-M(J,E(F,2,2))
6050 NEXT J
6060 P1=C(2)*D(3)-C(3)*D(2)
6070 P2=C(3)*D(1)-C(1)*D(3)
6080 P3=C(1)*D(2)-C(2)*D(1)
6090 Q1=1-M(1,E(F,1,2))
6100 Q2=1-M(2,E(F,1,2))
6110 Q3=500-M(3,E(F,1,2))
6120 W=P1*Q1+P2*Q2+P3*Q3
6130 IF W>=0 THEN GOSUB 2000
6140 NEXT F
6900 RETURN
6995 REM
7000 REM SHADING
7005 REM
7010 R1=M(1,E(F,2,1))-M(1,E(F,1,1))
7020 R2=M(2,E(F,2,1))-M(2,E(F,1,1))
7030 R3=M(3,E(F,2,1))-M(3,E(F,1,1))
7040 W1=SQR(R1*R1+R2*R2+R3*R3)
7050 R4=M(1,E(F,4,1))-M(1,E(F,1,1))
7060 R5=M(2,E(F,4,1))-M(2,E(F,1,1))
```

```
7070 R6=M(3,E(F,4,1))-M(3,E(F,1,1))
7080 W2=SQR(R4*R4+R5*R5+R6*R6)
7090 R1=R1/W1
7100 R2=R2/W1
7110 R3=R3/W1
7120 R4=R4/W2
7130 R5=R5/W2
7140 R6=R6/W2
7150 U=R3*R4-R1*R6
7160 IF U<=-.9 THEN RETURN
7170 IF U>-.9 AND U<-.5 THEN DS=8
7190 IF U>-.5 AND U<.1 THEN DS=6
7220 IF U>.1 AND U<.5 THEN DS=4
7240 IF U>.5 THEN DS=2
7270 FOR I=1 TO W1 STEP DS
7280 FOR Q=1 TO W2 STEP DS
7290 X=6*0.7*(M(1,E(F,1,1))+I*R1+Q*R4)
7300 Y=1023-6*(M(2,E(F,1,1))+I*R2+Q*R5)
7310 IF X<0 OR Y<0 THEN7340
7320 IF Y>1023 OR X>1023 THEN 7340
7330 POINT 3,X,Y
7340 NEXT Q
7350 NEXT I
7900 RETURN

READY.
```

THREE DIMENSIONAL SHAPE 4

DESCRIPTION

Perspective is that property of viewing an object which makes objects appear smaller the further away they are from the viewer. When looking down a long pole the pole appears to be tapered, but our understanding of the real world tells us that this is not so. Thus to add realism to a three dimensional computer display it is often desirable to add perspective to the display, this program is identical to the program THREE DIMENSIONAL SHAPE 1 except that an additional subroutine has been added to remove hidden lines, and the drawing routine has been modified to incorporate the hidden perspective algorithm.

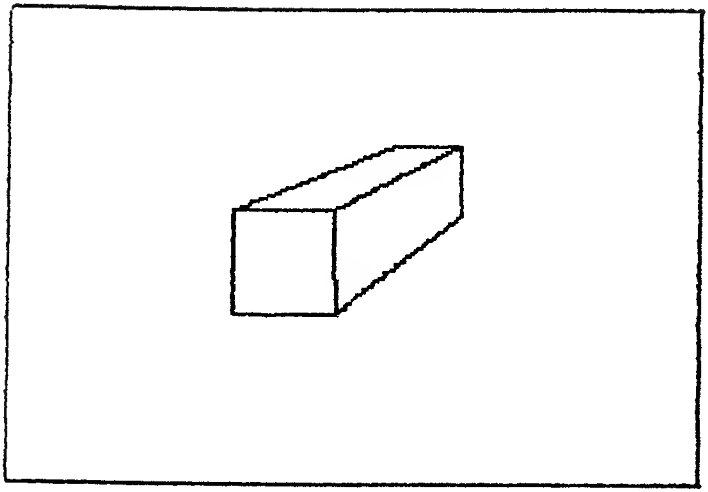
RUNNING THE PROGRAM

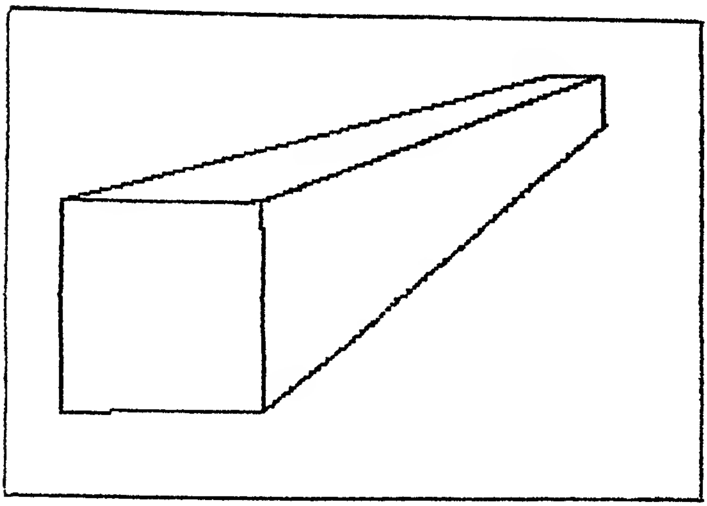
The parameters and data tables required by this program are the same as those used for the program THREE DIMENSIONAL SHAPE 2, consult this program for information.

PROGRAM STRUCTURE

Lines 1 to 5995 are identical to THREE DIMENSIONAL SHAPE 1 consult for details, except for the following:

2000-2240	shape drawing routine incorporating perspective algorithm in lines 2030 to 2045
6000-6140	subroutine to check for hidden surfaces





```
1 REM 3D DRAWING 4
2 REM ***********************
3 REM
10 REM A THREE DIMENSIONAL SHAPE IS DRAWN BY THIS
20 REM PROGRAM.THE ROTATION POSITION AND SCALE OF THE
30 REM OBJECT CAN BE CHANGED TO GIVE DIFFERENT VIEWING
35 REM ANGLES.THE PROGRAM INCORPORATES A ROUTINE TO
38 REM REMOVE HIDDEN LINES.THE OBJECT IS DISPLAYED WITH
39 REM PERSPECTIVE.
40 REM SET COLOURS
50 GRAPHIC 2
60 COLOR 3,3,0,10
65 REM
70 REM DRAW BORDER AROUND SCREEN
75 REM
80 GOSUB 900
85 REM
90 REM SET UP CONSTANTS, VARIABLES, AND ARRAYS
95 REM
100 DIM A(4,4)
110 DIM B(4,4)
115 DIM C(3)
117 DIM D(3)
120 SX=.1
130 SY=.1
140 SZ=.4
150 TX=-20
160 TY=-50
170 TZ=1
180 RX=1*π/180
190 RY=1*π/180
200 RZ=1*π/180
400 REM MAIN PROGRAM LOOP
410 GOSUB 1000
420 GOSUB 5000
430 GOSUB 3000
440 GOSUB 4000
450 GOSUB 6000
500 GET A$:IF A$="" THEN 500
510 COLOR 1,3,6,0
520 GRAPHIC 0
530 END
900 REM BORDER DRAWING SUBROUTINE
905 REM
910 POINT 3,0,0
920 DRAW 3 TO 0,1023
930 DRAW 3 TO 1023,1023
940 DRAW 3 TO 1023,0
950 DRAW 3 TO 0,0
960 RETURN
995 REM
```

```
1000 REM INITIALISE SHAPE
1005 REM
1010 NP=8
1020 NE=4
1025 NF=6
1030 REM
1040 DIM S(3,NP)
1050 DIM E(NF,NE,2)
1060 DIM M(3,NP)
1100 REM
1110 FOR N=1 TO NP
1120 READ S(1,N),S(2,N),S(3,N)
1130 NEXT N
1135 FOR F=1 TO NF
1140 FOR K=1 TO NE
1150 READ E(F,K,1),E(F,K,2)
1170 NEXT K
1180 NEXT F
1195 REM
1200 REM X,Y,Z POINT COORDINATES
1210 DATA 0,0,200,200,0,200,200,0,0,0,0,0
1220 DATA 0,200,200,200,200,200,200,200,0,0,200,0
1295 REM
1300 REM CONNECTION DATA
1305 REM
1310 DATA 1,2,2,3,3,4,4,1
1320 DATA 5,1,1,4,4,8,8,5
1330 DATA 6,5,5,8,8,7,7,6
1340 DATA 2,6,6,7,7,3,3,2
1350 DATA 1,5,5,6,6,2,2,1
1360 DATA 3,7,7,8,8,4,4,3
1900 RETURN
1995 REM
2000 REM DRAW SHAPE WITH PERSPECTIVE
2005 REM
2020 FOR K=1 TO NE
2030 V1=E(F,K,1)
2033 PE=ABS(300/(M(3,V1)-300))
2040 V2=E(F,K,2)
2043 PF=ABS(300/(M(3,V2)-300))
2045 IF V1=0 THEN 2240
2050 XB=PE*M(1,V1)
2060 YB=PE*M(2,V1)
2070 XE=PF*M(1,V2)
2080 YE=PF*M(2,V2)
2090 DS=1
2100 P=XE-XB
2110 Q=YE-YB
2120 R=SQR(P*P+Q*Q)
2130 LX=P/R
2140 LY=Q/R
```

```
2150 FOR I=0 TO R STEP DS
2160 X=6*0.7*(XB+I*LX)
2170 Y=1023-6*(YB+I*LY)
2180 IF X<0 OR Y<0 THEN 2230
2190 IF X>1023 OR Y>1023 THEN 2230
2220 POINT 3,X,Y
2230 NEXT I
2240 NEXT K
2900 RETURN
2995 REM
3000 REM SET TRANSFORMATION MATRIX
3005 REM
3010 A(1,1)=COS(RY)*COS(RZ)
3020 A(1,2)=COS(RY)*SIN(RZ)
3030 A(1,3)=-SIN(RY)
3040 A(1,4)=0
3050 A(2,1)=COS(RX)*(-SIN(RZ))+SIN(RX)*SIN(RY)*COS(RZ)
3060 A(2,2)=COS(RX)*COS(RZ)+SIN(RX)*SIN(RY)*SIN(RZ)
3070 A(2,3)=SIN(RX)*COS(RY)
3080 A(2,4)=0
3090 A(3,1)=(-SIN(RX))*(-SIN(RZ))+COS(RX)*SIN(RY)*COS(RZ
3100 A(3,2)=-SIN(RX)*COS(RZ)+COS(RZ)*SIN(RY)*SIN(RZ)
3110 A(3,3)=COS(RX)*COS(RY)
3120 A(3,4)=0
3130 A(4,1)=0
3140 A(4,2)=0
3150 A(4,3)=0
3160 A(4,4)=1
3195 REM
3200 REM SET UP SCALING AND TRANSLATION MATRIX
3205 REM
3210 B(1,1)=SX*A(1,1)
3220 B(1,2)=SX*A(1,2)
3230 B(1,3)=SX*A(1,3)
3240 REM
3250 B(2,1)=SY*A(2,1)
3260 B(2,2)=SY*A(2,2)
3270 B(2,3)=SY*A(2,3)
3280 REM
3290 B(3,1)=SZ*A(3,1)
3300 B(3,2)=SZ*A(3,2)
3310 B(3,3)=SZ*A(3,3)
3320 REM
3330 B(4,1)=TX
3340 B(4,2)=TY
3350 B(4,3)=TZ
3900 RETURN
3995 REM
4000 REM PERFORM TRANSLATION
4005 REM
4010 FOR Q=1 TO NP
```

```
4015 REM
4020 XT=S(1,Q)-XC
4030 YT=S(2,Q)-YC
4040 ZT=S(3,Q)-ZC
4045 REM
4050 M(1,Q)=XC+(XT*B(1,1)+YT*B(2,1)+ZT*B(3,1)+B(4,1))
4060 M(2,Q)=YC+(XT*B(1,2)+YT*B(2,2)+ZT*B(3,2)+B(4,2))
4070 M(3,Q)=ZC+(XT*B(1,3)+YT*B(2,3)+ZT*B(3,3)+B(4,3))
4080 NEXT Q
4900 RETURN
4995 REM
5000 REM FIND CENTROID
5005 REM
5010 P=0:Q=0:R=0
5020 FOR I=1 TO NP
5030 P=P+S(1,I)
5040 Q=Q+S(2,I)
5050 R=R+S(3,I)
5060 NEXT I
5070 XC=P/NP
5080 YC=Q/NP
5090 ZC=R/NP
5900 RETURN
5995 REM
6000 REM HIDDEN SURFACE CHECK
6005 REM
6010 FOR F=1 TO NF
6020 FOR J=1 TO 3
6030 C(J)=M(J,E(F,1,2))-M(J,E(F,1,1))
6040 D(J)=M(J,E(F,2,1))-M(J,E(F,2,2))
6050 NEXT J
6060 P1=C(2)*D(3)-C(3)*D(2)
6070 P2=C(3)*D(1)-C(1)*D(3)
6080 P3=C(1)*D(2)-C(2)*D(1)
6090 Q1=1-M(1,E(F,1,2))
6100 Q2=1-M(2,E(F,1,2))
6110 Q3=500-M(3,E(F,1,2))
6120 W=P1*Q1+P2*Q2+P3*Q3
6130 IF W>=0 THEN GOSUB 2000
6140 NEXT F
6900 RETURN
```

INDEX

Arc1 71

Big Character 112

Character Building 106
Circle 63
Colour Control 6
Colours 26
Computer Art 5

Disk 1 75

Ellipse 67

Fan 22

Graph 88

Hidden Lines 167
High Resolution 30
Hi-Res Cursor 100
Hi-Res Cursor 2 101

Interpolate 95

Line 44

Map 16
Move 153

Perspective 180
Piechart 83
Polygon 1 51
Polygon 2 54

Rainbow 19
Random Colours 14
Rectangle 1 48
Rectangle 2 58
Rotate 138
Rotate 2 142
Rotate 3 147

Scale 1 118
Scale 2 122
Segment 79
Shading 173
Stretch 1 126
Stretch 2 131

Three Dimension Graph 92
Three Dimension 1 160
Three Dimension 2 167
Three Dimension 3 173
Three Dimension 4 180

Duckworth Personal Computing series

VIC Graphics

Written by Nick Hampshire, author of many books on popular computing and publisher of *Commodore Computing International*, this book provides the reader with an introduction to programming techniques used to generate graphics displays on a Commodore VIC. Topics covered include:

- Using colour
- Two dimensional shape plotting
- Shape plotting
- Shape scaling and stretching
- Shape movement
- Shape rotation
- Plotting using matrix manipulation
- Three dimensional shape plotting

Vic Graphics is a must for every VIC user who wishes to use the machine to its maximum graphics display potential.

Duckworth
The Old Piano Factory
43 Gloucester Crescent, London NW1

ISBN 0 7156 1702 8

IN UK ONLY £6.95 NET